

---

---

Harlequin Dylan

# Dylan User Interface Manager

## Library Reference

Version 2.0 Beta



## **Copyright and Trademarks**

*Harlequin Dylan: Library Reference: Dylan User Interface Manager*

Version 2.0 Beta

May 1999

Part number: 3DPDTZ-15ME

Copyright © 1998–1999 by Harlequin Group plc.

Companies, names and data used in examples herein are fictitious unless otherwise noted.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Harlequin Group plc.

The information in this publication is provided for information only and is subject to change without notice. Harlequin Group plc and its affiliates assume no responsibility or liability for any loss or damage that may arise from the use of any information in this publication. The software described in this book is furnished under license and may only be used or copied in accordance with the terms of that license.

Dylan is a trademark of Apple Computer, Inc.

Harlequin is a trademark of Harlequin Group plc.

Other brand or product names are the registered trademarks or trademarks of their respective holders.

### **US Government Use**

The Dylan Software is a computer software program developed at private expense and is subject to the following Restricted Rights Legend: "Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in (i) FAR 52.227-14 Alt III or (ii) FAR 52.227-19, as applicable. Use by agencies of the Department of Defense (DOD) is subject to Harlequin's customary commercial license as contained in the accompanying license agreement, in accordance with DFAR 227.7202-1(a). For purposes of the FAR, the Software shall be deemed to be 'unpublished' and licensed with disclosure prohibitions, rights reserved under the copyright laws of the United States. Harlequin Incorporated, One Cambridge Center, Cambridge, Massachusetts 02142."

<http://www.harlequin.com/>

#### **Europe:**

##### **Harlequin Limited**

Barrington Hall  
Barrington  
Cambridge CB2 5RG  
UK

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

#### **North America:**

##### **Harlequin Incorporated**

One Cambridge Center  
Cambridge, MA 02142  
USA

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

#### **Asia Pacific:**

##### **Harlequin Australia Pty. Ltd.**

Level 12  
12 Moore Street  
Canberra, ACT 2601  
Australia

telephone +61 2 6206 5522  
fax +61 2 6206 5525

---

---

# Contents

## 1 Conventions in this Manual 1

Audience, goals, and purpose	1
Example code fragments	2
Module structure	2
Spread point arguments to functions	3
Immutability of objects	3
Specialized arguments to generic functions	5
Macros that expand into calls to advertised functions	5
Terminology pertaining to error conditions	6

## 2 DUIM-Geometry Library 7

Overview	7
The class hierarchy for DUIM-Geometry	7
DUIM-Geometry Module	9

## 3 DUIM-Extended-Geometry Library 67

Overview	67
The class hierarchy for DUIM-Extended-Geometry	67
DUIM-Extended-Geometry Module	68

## 4 DUIM-DCs Library 101

Overview	101
The class hierarchy for DUIM-DCs	102
DUIM-DCs Module	104

<b>5</b>	<b>DUIM-Sheets Library</b>	<b>173</b>
	Overview	173
	The class hierarchy for DUIM-Sheets	176
	DUIM-Sheets Module	183
<b>6</b>	<b>DUIM-Graphics Library</b>	<b>339</b>
	Overview	339
	Definitions	340
	Drawing is approximate	341
	Rendering conventions for geometric shapes	343
	Drawing using path related functions	350
	DUIM-Graphics Module	354
<b>7</b>	<b>DUIM-Layouts Library</b>	<b>395</b>
	Overview	395
	The class hierarchy for DUIM-Layouts	395
	DUIM-Layouts Module	399
<b>8</b>	<b>DUIM-Gadgets Library</b>	<b>445</b>
	Overview	445
	Callbacks and keys	447
	Gadget protocols	448
	The class hierarchy for DUIM-Gadgets	449
	Button gadgets	459
	Text gadgets	459
	Collection gadgets	460
	Value range gadgets	462
	Page gadgets	462
	Gadgets that can have children	463
	DUIM-Gadgets Module	465
<b>9</b>	<b>DUIM-Frames Library</b>	<b>615</b>
	Overview	615
	The class hierarchy for DUIM-Frames	616
	DUIM-Commands Library	620
	DUIM-Frames Module	621

# 1

---

---

# Conventions in this Manual

This chapter describes the conventions used in this manual and in the DUIM software itself.

## 1.1 Audience, goals, and purpose

This manual is intended for programmers using DUIM, and forms a complete reference for the Application Programmer’s Interface (API) for DUIM. You should also see *Building Applications using DUIM* for a description of how to start building applications using DUIM. At some points, the API also includes lower-level layers, which DUIM programmers are free to specialize.

The DUIM library is a set of interfaces that allow you to create graphical user interfaces (GUIs) for your application using Dylan code.

In this document, we may refer to two different audiences. A *user* is a person who uses an application program that was written using DUIM. A *DUIM programmer* is a person who writes application programs using DUIM. Generally, this manual assumes that you, the reader, are the programmer.

## 1.2 Example code fragments

Throughout this manual, example code fragments are provided at suitable points in the documentation. These provide short illustrations of how to use the interfaces being described. If you wish, you can run these examples interactively by typing them into the Dylan Playground.

A number of additional, longer examples are provided as part of the Harlequin Dylan installation, and are installed on your hard disk automatically. You can look at these examples and load them into the environment by clicking on the examples button in the main window of the Harlequin Dylan environment.

Longer examples are also provided and discussed fully in the *Building Applications using DUIM*, which you should refer to for an introduction to building DUIM applications.

## 1.3 Module structure

The functionality of DUIM is provided via a number of modules. Each chapter of this manual indicates what module its API is exported from.

The `duim` module is the main API module, which contains the variables for the API-level functions available.

The `duim-geometry` module provides basic support for coordinate geometry. This allows the position of elements in a window object to be determined correctly.

The `duim-extended-geometry` module provides more extensive support for co-ordinate geometry that is only required for more specialist uses.

The `duim-dcs` module provides color support to the DUIM library.

The `duim-sheets` module provides basic support for sheets. Sheets are the basic unit of window applications, and can be nested hierarchically to build up a complete user interface.

The `duim-graphics` module provides support for graphics drawing

The `duim-layouts` module provides support for a layout protocol that makes it easy to create and layout groups of related elements in a given interface. This module can handle layout problems such as the spacing and justification of a group of elements automatically.

The `duim-gadgets` module provides all the gadgets available for use in the DUIM library. Gadgets are the sheet objects that make up any user interface, and the DUIM library supplies all the gadgets you will need in your applications.

The `duim-frames` module provides support for frames. A DUIM frame is a combination of a set of nested sheets, together with an event loop that describes the behavior of the elements in those sheets. DUIM frames can be used to specify whether a given user interface is displayed in an application as a dialog box, or a more straightforward window, or as a task wizard, and so on.

The Dylan Playground should be used when you just want to experiment with DUIM code fragments without creating modules of your own. For real application code, of course, you should define your own modules and libraries and use the appropriate library code required by your application.

## 1.4 Spread point arguments to functions

Many functions that take point arguments come in two forms: *structured* and *spread*. Functions that take structured point arguments take the argument as a single `point` object. Functions that take spread point arguments take a pair of arguments that correspond to the `x` and `y` coordinates of the point.

Functions that take structured point arguments, or return structured point values have an asterisk in their name, for example, `draw-line*`.

## 1.5 Immutability of objects

Most DUIM objects are *immutable*, that is, at the API level none of their components can be modified once the object is created. Examples of immutable objects include all of the members of the `<region>` classes, pens, brushes, colors, and text styles. Since immutable objects by definition never change, functions in the DUIM API can safely capture immutable objects without first

copying them. This also allows DUIM to cache immutable objects. Any `make` methods that return immutable objects are free to either create and return a new object, or return an already existing object.

A few DUIM objects are *mutable*. Some components of mutable objects can be modified once the object has been created, usually via setter functions.

In DUIM, object immutability is maintained at the class level. Throughout this specification, the immutability or mutability of a class will be explicitly specified.

Some immutable classes also allow *interning*. A class is said to be interning if it guarantees that two instances that are equivalent will always be `==`. For example, the class `<text-style>` is interned, so calling `make-text-style` twice with the same arguments would return identical values.

In some rare cases, DUIM *will* modify objects that are members of immutable classes. Such objects are referred to as being *volatile*. Extreme care must be taken with volatile objects. For example, objects of class `<bounding-box>` are often volatile.

### 1.5.1 Behavior of interfaces

Any interfaces that take or return mutable objects can be classified in a few different ways.

Most functions *do not capture* their mutable input objects, that is, these functions will either not store the objects at all, or will copy any mutable objects before storing them, or perhaps store only some of the components of the objects. Later modifications to those objects will not affect the internal state of DUIM.

Some functions *may capture* their mutable input objects. That is, it is not specified whether the mutable inputs to these functions will or will not be captured. For such functions, you should assume that these objects will be captured and must not modify these objects capriciously. Furthermore, the behavior is undefined if these objects are later modified.

Some functions that return mutable objects are guaranteed to create *fresh outputs*. These objects can be modified without affecting the internal state of DUIM.

Functions that return mutable objects that are not fresh objects fall into two categories:

- Those that return *read-only state*
- Those that return *read/write state*

If a function returns read-only state, programmers must not modify that object; doing so might corrupt the state of DUIM. If a function returns read/write state, the modification of that object is part of the DUIM interface, and you are free to modify the object in ways that make sense.

## 1.6 Specialized arguments to generic functions

Unless otherwise stated, this manual uses the following convention for specifying which arguments to generic functions are specialized:

- If the generic function is a `-setter` function, the second argument is the one that is intended to be specialized.
- If the generic function is a “mapping” function (such as `do-sheets`), the second argument (the object that specifies what is being mapped over) is the one that is specialized. The first argument (the functional argument) is not intended to be specialized.
- Otherwise, the first argument is the one that is intended to be specialized.

## 1.7 Macros that expand into calls to advertised functions

Many macros that take a “body” argument expand into a call to an advertised function that takes a functional argument. This functional argument will execute the supplied body. For a macro named `with-environment`, the function is generally named `do-with-environment`. For example, `with-drawing-options` might be defined as follows:

```
define macro with-drawing-options
  { with-drawing-options
    (?medium:name, #rest ?keys:*) ?body:body end }
    => { begin
      let with-drawing-options-body =
        method (?medium) ?body end;
        do-with-drawing-options(?medium,
          with-drawing-options-body, ?keys)
      end }
  end macro;

define method do-with-drawing-options
  (medium :: <medium>, function, #rest options)
  apply(merge-drawing-options-into-medium, medium, options);
  function(medium)
end;
```

## 1.8 Terminology pertaining to error conditions

When this documentation specifies that it “is an error” for some situation to occur, this means that:

- No valid DUIM program should cause this situation to occur.
- If this situation does occur, the effects and results are undefined.
- DUIM often tries to detect such an error, but it might not.

When this manual specifies that some argument “must be a *type*” or uses the phrase ``the *type* argument”, this means that it is an error if the argument is not of the specified *type*. DUIM tries to detect such type errors, but it might not always be successful.

When this documentation says that “an error is signalled” in some situation, this means that:

- If the situation occurs, DUIM will signal an error using `error` or `cerror`.
- Valid DUIM programs may rely on the fact that an error will be signalled.

When this manual states that “a condition is signalled” in a given situation, this is the same as saying that “an error is signalled”, with the exception that the condition will be signalled using `signal` instead of `error`.

# 2

---

---

# DUIM-Geometry Library

## 2.1 Overview

The DUIM-Geometry library provides basic support for coordinate geometry. This allows the position of elements in a window object to be determined correctly. The library contains a single module, `duim-geometry`, from which all the interfaces described in this chapter are exposed. Section 2.3 on page 9 contains complete reference entries for each exposed interface.

## 2.2 The class hierarchy for DUIM-Geometry

The base classes for classes in the DUIM-Geometry library are `<region>` and `<transform>`, both of which are subclasses of `<object>`. While the `<region>` class has a number of subclasses, `<transform>` has no direct subclasses.

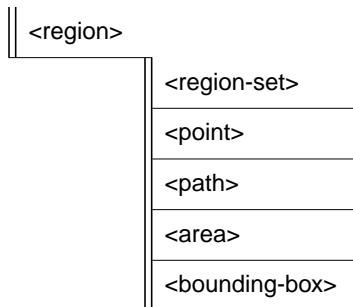
`<transform>` The superclass of all transforms. A transform describes the mapping of one set of points onto another. There are one or more subclasses of `<transform>` that implement transforms. These subclasses have implementation-dependent names which are explicitly unspecified. All of the instantiable transformation classes provided by DUIM are immutable.

In addition, there are a number of error classes which may be signalled. These are all subclasses of `<error>`.

### 2.2.1 The `<region>` class and its subclasses

The DUIM-Geometry library exposes the `<region>` class and its subclasses as shown in Table 2.1. None of these subclasses have any further subclasses exposed in the DUIM-Geometry library, although the DUIM-Extended-Geometry library exposes some subclasses of `<area>` and `<path>`.

**Table 2.1** The `<region>` class and its subclasses

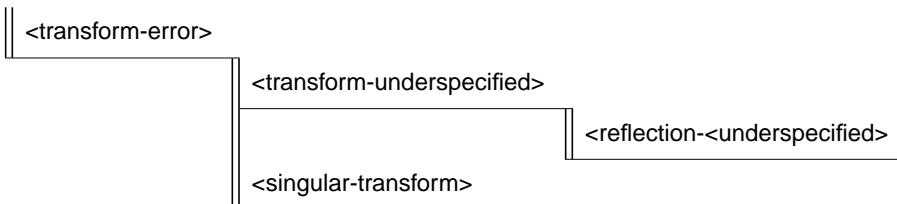


<code>&lt;region&gt;</code>	This class is used to represent any set of points. The <code>&lt;region&gt;</code> class includes both bounded regions (that is, regions whose edges are known) and unbounded regions (that is, regions with no known edges).
<code>&lt;region-set&gt;</code>	This class represents a region set, that is, a set of regions.
<code>&lt;point&gt;</code>	This class is used to represent mathematical points (that is, regions with dimensionality 0).
<code>&lt;path&gt;</code>	The class <code>&lt;path&gt;</code> denotes bounded regions with a length, but no area (that is, they have dimensionality 1).
<code>&lt;area&gt;</code>	This class denotes bounded regions that have an area (that is, they have dimensionality 2).
<code>&lt;bounding-box&gt;</code>	A bounding box is an axis aligned rectangle that contains some region.

## 2.2.2 Error classes provided by DUIM-Geometry

The DUIM-Geometry library exposes a number of errors that can be signalled in certain circumstances. They are shown in Table 2.2. All the errors shown are subclasses of the `<error>` class. Note that the subclasses of `<transform-error>` are all specific to particular errors.

**Table 2.2** The `<transform-error>` class and its subclasses



### `<transform-error>`

The superclass of all error conditions signalled when there is an error with a transform.

### `<transform-underspecified>`

The error that is signalled when `make-3-point-transform` is given three colinear image points.

### `<reflection-<underspecified>`

The error that is signalled when `make-reflection-transform` is given two coincident points.

### `<singular-transform>`

The error that is signalled when `invert-transform` is called on a singular transform, that is, a transform that has no inverse.

## 2.3 DUIM-Geometry Module

This section contains a complete reference of all the interfaces that are exported from the `duim-geometry` module.

=		<i>G.f. method</i>
Summary	Tests if its arguments are equal.	
Signature	= <i>region1 region2 =&gt; boolean</i> = <i>transform1 transform2 =&gt; boolean</i>	
Arguments	<i>region1</i> An instance of type < <b>region</b> >. <i>region2</i> An instance of type < <b>region</b> >. <i>transform1</i> An instance of type < <b>transform</b> >. <i>transform2</i> An instance of type < <b>transform</b> >.	
Values	<i>boolean</i> An instance of type < <b>boolean</b> >.	
Description	Tests if its arguments are equal. Returns #t if the two regions or transforms are the same, otherwise returns #f. Two regions are considered equal if they contain exactly the same set of points. Two transforms are considered equal if they transform every region the same way.	

< <b>area</b> >	<i>Open abstract class</i>
Summary	The class < <b>area</b> > denotes bounded regions that have dimensionality 2 (that is, have area).
Superclasses	< <b>region</b> >
Init-keywords	None.
Description	The class < <b>area</b> > denotes bounded regions that have dimensionality 2 (that is, have area).< <b>area</b> > is a subclass of < <b>region</b> >.

Note that constructing an area object with no area (such as calling `make-rectangle` with two coincident points, for example) may canonicalize it to `$nowhere`.

Operations	The following operation is exported from the <b>DUIM-Geometry</b> module.
	<code>area?</code>
See also	<code>area?</code> , page 11

## **area?** *Generic function*

Summary	Returns <code>#t</code> if its argument is an area, otherwise returns <code>#f</code> .	
Signature	<code>area? object =&gt; boolean</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if <code>object</code> is an area, otherwise returns <code>#f</code>	
See also	<code>&lt;area&gt;</code> , page 10	

## **<bounding-box>** *Open abstract instantiable class*

Summary	The class that represents a bounding box.	
Superclasses	<code>&lt;region&gt;</code>	
Init-keywords	<code>left:</code>	An instance of type <code>&lt;integer&gt;</code> .
	<code>top:</code>	An instance of type <code>&lt;integer&gt;</code> .
	<code>right:</code>	An instance of type <code>&lt;integer&gt;</code> .

	<b>bottom:</b> An instance of type <integer>.
Description	A bounding box is an axis aligned rectangle that contains some region. The representation of bounding boxes in DUIM is chosen to be efficient. This representation is not sufficient to represent the result of arbitrary transformations (such as rotations) of bounding boxes. The most general class of transformations that is guaranteed to transform a box into another box is the class of transformations that satisfy <code>rectilinear-transformation?</code> .  Bounding boxes are immutable, but since they reflect the live state of such mutable objects as sheets, bounding boxes are volatile. Therefore, programmers must not depend on the bounding box associated with a mutable object remaining constant.
Operations	The following operations are exported from the <code>DUIM-Geometry</code> module.  <code>bounding-box? box-edges region-contains-position? region-contains-region? region-difference region-empty? region-intersection region-intersects-region? region-union set-box-edges set-box-position set-box-size transform-region untransform-region</code>
See also	<code>bounding-box?</code> , page 12 <code>bounding-box</code> , page 13 <code>box-edges</code> , page 14

<b>bounding-box?</b> <i>Generic function</i>	
Summary	Returns true if its argument is a bounding box.
Signature	<code>bounding-box? object =&gt; boolean</code>

Arguments	<i>object</i>	An instance of type < <code>object</code> >.
Values	<i>boolean</i>	An instance of type < <code>boolean</code> >.
Description		Returns <code>#t</code> if <i>object</i> is a bounding box (that is, supports the bounding box protocol), otherwise returns <code>#f</code> .
See also		<a href="#">&lt;bounding-box&gt;</a> , page 11 <a href="#">bounding-box</a> , page 13 <a href="#">box-edges</a> , page 14

## bounding-box *Generic function*

Summary	Returns the bounding box of a region.	
Signature	<code>bounding-box region #key into =&gt; box</code>	
Arguments	<i>region</i>	An instance of type < <code>region</code> >.
	<i>into</i>	An instance of type <code>false-or(&lt;bounding-box&gt;)</code> .
Values	<i>box</i>	An instance of type < <code>bounding-box</code> >.
Description	<p>The argument <i>region</i> must be either a bounded region (such as a line or an ellipse) or some other object that obeys the bounding box protocol, such as a sheet.</p> <p>This function often returns an existing object, so you should not modify the returned result.</p> <p>If <i>into</i> is supplied, it is a bounding box that might be destructively modified to contain the result.</p>	
See also	<a href="#">&lt;bounding-box&gt;</a> , page 11 <a href="#">bounding-box?</a> , page 12	

**box-bottom**, page 14

	<i>Function</i>
Summary	Returns the y coordinate of the bottom right corner of the bounding box of a region.
Signature	<b>box-bottom</b> <i>region</i> => <i>bottom</i>
Arguments	<i>region</i> An instance of type < <i>region</i> >.
Values	<i>bottom</i> An instance of type < <i>integer</i> >.
Description	Returns the y coordinate of the bottom right corner of the bounding box of <i>region</i> . The argument <i>region</i> must be either a bounded region or some other object that obeys the bounding box protocol.
See also	<a href="#">box-left</a> , page 16 <a href="#">box-right</a> , page 17 <a href="#">box-top</a> , page 18

	<i>Generic function</i>
Summary	Returns the bounding box of a region.
Signature	<b>box-edges</b> <i>region</i> => <i>left top right bottom</i>
Arguments	<i>region</i> An instance of type < <i>region</i> >.
Values	<i>left</i> An instance of type < <i>integer</i> >. <i>top</i> An instance of type < <i>integer</i> >. <i>right</i> An instance of type < <i>integer</i> >.

	<i>bottom</i>	An instance of type <integer>.
Description	Returns the bounding box of <i>region</i> as four integers specifying the <i>x</i> and <i>y</i> coordinates of the top left point and the <i>x</i> and <i>y</i> coordinates of the bottom right point of the box	
	The argument <i>region</i> must be either a bounded region (such as a line or an ellipse) or some other object that obeys the bounding box protocol, such as a sheet.	
	The four returned values <i>left</i> , <i>top</i> , <i>right</i> , and <i>bottom</i> will satisfy the inequalities	
	<i>left</i> $\leq$ <i>right</i> <i>top</i> $\leq$ <i>bottom</i>	
See also	<bounding-box>, page 11 bounding-box?, page 12 bounding-box, page 13	

		<i>Function</i>
Summary	Returns the height of the bounding box of a region.	
Signature	<b>box-height</b> <i>region</i> => <i>height</i>	
Arguments	<i>region</i>	An instance of type <region>.
Values	<i>height</i>	An instance of type <integer>.
Description	Returns the height of the bounding box <i>region</i> . The height of a bounding box is the difference between the maximum <i>y</i> coordinate and its minimum <i>y</i> coordinate. The argument <i>region</i> must be either a bounded region or some other object that obeys the bounding box protocol.	
See also	<b>box-position</b> , page 16	

**box-size**, page 18

**box-width**, page 19

## box-left *Function*

Summary	Returns the <i>x</i> coordinate of the upper left corner of the bounding box of a region.	
Signature	<b>box-left</b> <i>region</i> => <i>left</i>	
Arguments	<i>region</i>	An instance of type < <i>region</i> >.
Values	<i>left</i>	An instance of type < <i>integer</i> >.
Description	Returns the <i>x</i> coordinate of the upper left corner of the bounding box <i>region</i> . The argument <i>region</i> must be either a bounded region or some other object that obeys the bounding box protocol, such as a sheet.	
See also	<a href="#">box-bottom</a> , page 14 <a href="#">box-right</a> , page 17 <a href="#">box-top</a> , page 18	

## box-position *Generic function*

Summary	Returns the position of the bounding box of a region as two values.	
Signature	<b>box-position</b> <i>region</i> => <i>x</i> <i>y</i>	
Arguments	<i>region</i>	An instance of type < <i>region</i> >.
Values	<i>x</i>	An instance of type < <i>integer</i> >.

*y*

Description	Returns the position of the bounding box of <i>region</i> as two values. The position of a bounding box is specified by its top left point.
See also	<a href="#">box-height, page 15</a> <a href="#">box-size, page 18</a> <a href="#">box-width, page 19</a>

## **box-right** *Function*

Summary	Returns the <i>x</i> coordinate of the bottom right corner of the bounding box of a region.	
Signature	<b>box-right</b> <i>region</i> => <i>right</i>	
Arguments	<i>region</i>	An instance of type <region>.
Values	<i>right</i>	An instance of type <integer>.
Description	Returns the <i>x</i> coordinate of the bottom right corner of the bounding box <i>region</i> . The argument <i>region</i> must be either a bounded region or some other object that obeys the bounding box protocol, such as a sheet.	
See also	<a href="#">box-bottom, page 14</a> <a href="#">box-left, page 16</a> <a href="#">box-top, page 18</a>	

		<i>Generic function</i>
Summary	Returns the width and height of the bounding box of a region as two values	
Signature	<b>box-size</b> <i>region</i> => <i>width height</i>	
Arguments	<i>region</i>	An instance of type < <i>region</i> >.
Values	<i>width</i>	An instance of type < <i>integer</i> >.
	<i>height</i>	An instance of type < <i>integer</i> >.
Description	Returns the width and height of the bounding box of <i>region</i> as two values. The argument <i>region</i> must be either a bounded region or some other object that obeys the bounding box protocol, such as a sheet.	
See also	<b>box-height</b> , page 15 <b>box-position</b> , page 16 <b>box-width</b> , page 19	

		<i>Function</i>
Summary	Returns the y coordinate of the upper left corner of the bounding box of a region.	
Signature	<b>box-top</b> <i>region</i> => <i>top</i>	
Arguments	<i>region</i>	An instance of type < <i>region</i> >.
Values	<i>top</i>	An instance of type < <i>integer</i> >.

**Description** Returns the *y* coordinate of the upper left corner of the bounding box *region*. The argument *region* must be either a bounded region or some other object that obeys the bounding box protocol.

**See also**

- `box-bottom`, page 14
- `box-left`, page 16
- `box-right`, page 17

## **box-width** *Function*

**Summary** Returns the width of the bounding box of a region.

**Signature** `box-width region => width`

**Arguments** `region` An instance of type `<region>`.

**Values** `width` An instance of type `<integer>`.

**Description** Returns the width of the bounding box *region*. The width of a bounding box is the difference between its maximum *x* coordinate (right) and its minimum *x* coordinate (left). The argument *region* must be either a bounded region or some other object that obeys the bounding box protocol, such as a sheet.

**See also**

- `box-height`, page 15
- `box-position`, page 16
- `box-size`, page 18

## **compose-rotation-with-transform** *Generic function*

**Summary** Creates a new transform by composing a transform with the given rotation

Signature	<code>compose-rotation-with-transform transform angle #key origin =&gt; transform</code>	
Arguments	<code>transform</code>	An instance of type <code>&lt;transform&gt;</code> .
	<code>angle</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>origin</code>	An instance of type <code>&lt;point&gt;</code> . Default value: $(0, 0)$ .
Values	<code>transform</code>	An instance of type <code>&lt;transform&gt;</code> .
Description	<p>Creates a new transform by composing the transform <code>transform</code> with the given rotation. The order of composition is that the rotation transform is applied first, followed by the argument <code>transform</code>.</p> <p>Note that this function could be implemented by using <code>make-rotation-transform</code> and <code>compose-transforms</code>. It is provided because it is common to build up a transform as a series of simple transforms.</p>	
See also	<code>make-rotation-transform</code> , page 35	

## compose-scaling-with-transform *Generic function*

Summary	Creates a new transform by composing a transform with the given scaling.	
Signature	<code>compose-scaling-with-transform transform scale-x scale-y #key origin =&gt; transform</code>	
Arguments	<code>transform</code>	An instance of type <code>&lt;transform&gt;</code> .
	<code>scale-x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>scale-y</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>origin</code>	An instance of type <code>&lt;point&gt;</code> . Default value: $(0, 0)$ .

Values	<i>transform</i>	An instance of type < <code>transform</code> >.
Description	<p>Creates a new transform by composing the transform <i>transform</i> with the given scaling. The order of composition is that the scaling transform is applied first, followed by the argument <i>transform</i>.</p> <p>The argument <i>scale-x</i> represents the scaling factor for the <i>x</i> direction.</p> <p>The argument <i>scale-y</i> represents the scaling factor for the <i>y</i> direction.</p> <p>The argument <i>origin</i> represents the point around which scaling is performed. The default is to scale around the origin.</p> <p>Note that this function could be implemented by using <code>make-scaling-transform</code> and <code>compose-transforms</code>. It is provided because it is common to build up a transform as a series of simple transforms.</p>	
See also	<code>make-scaling-transform</code> , page 36	

<b>compose-transforms</b>		<i>Generic function</i>
Summary	Returns a transform that is the mathematical composition of its arguments.	
Signature	<code>compose-transforms transform1 transform2 =&gt; transform</code>	
Arguments	<i>transform1</i>	An instance of type < <code>transform</code> >.
	<i>transform2</i>	An instance of type < <code>transform</code> >.
Values	<i>transform</i>	An instance of type < <code>transform</code> >.

Description	Returns a transform that is the mathematical composition of its arguments. Composition is in right-to-left order, that is, the resulting transform represents the effects of applying the transform <i>transform2</i> followed by the transform <i>transform1</i> .
See also	<code>compose-transform-with-rotation</code> , page 22

## **compose-transform-with-rotation** *Generic function*

Summary	Creates a new transform by composing a given rotation with a transform.	
Signature	<code>compose-transform-with-rotation transform angle #key origin =&gt; transform</code>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
	<i>angle</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>origin</i>	An instance of type <code>&lt;point&gt;</code> . Default value: (0,0).
Values	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
Description	<p>Creates a new transform by composing a given rotation with the transform <i>transform</i>. The order of composition is <i>transform</i> first, followed by the rotation transform.</p> <p>The argument <i>angle</i> represents the angle by which to rotate, in radians.</p> <p>The argument <i>origin</i> represents the point about which to rotate. The default is to rotate around (0,0).</p> <p>Note that this function could be implemented by using <code>make-rotation-transform</code> and <code>compose-transforms</code>. It is provided because it is common to build up a transform as a series of simple transforms.</p>	

See also [compose-transforms](#), page 21

[make-rotation-transform](#), page 35

## compose-transform-with-scaling *Generic function*

**Summary** Creates a new transform by composing a given scaling with a transform.

**Signature** `compose-transform-with-scaling transform scale-x scale-y  
#key origin => transform`

**Arguments**

<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
<i>scale-x</i>	An instance of type <code>&lt;real&gt;</code> .
<i>scale-y</i>	An instance of type <code>&lt;real&gt;</code> .
<i>origin</i>	An instance of type <code>&lt;point&gt;</code> . Default value: (0,0).

**Values** *transform* An instance of type `<transform>`.

**Description** Creates a new transform by composing a given scaling with the transform *transform*. The order of composition is *transform* first, followed by the scaling transform.

The argument *scale-x* represents the scaling factor for the *x* direction.

The argument *scale-y* represents the scaling factor for the *y* direction.

The argument *origin* represents the point around which scaling is performed. The default is to scale around the origin.

Note that this function could be implemented by using `make-scaling-transform` and `compose-transforms`. It is provided because it is common to build up a transform as a series of simple transforms.

See also [compose-transforms](#), page 21.  
[make-scaling-transform](#), page 36

## compose-transform-with-translation      *Generic function*

Summary	Creates a new transform by composing a given translation with a transform.						
Signature	<code>compose-transform-with-translation <i>transform</i> <i>dx</i> <i>dy</i> =&gt; <i>transform</i></code>						
Arguments	<table><tr><td><i>transform</i></td><td>An instance of type <code>&lt;transform&gt;</code>.</td></tr><tr><td><i>dx</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr><tr><td><i>dy</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr></table>	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .	<i>dx</i>	An instance of type <code>&lt;real&gt;</code> .	<i>dy</i>	An instance of type <code>&lt;real&gt;</code> .
<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .						
<i>dx</i>	An instance of type <code>&lt;real&gt;</code> .						
<i>dy</i>	An instance of type <code>&lt;real&gt;</code> .						
Values	<i>transform</i> An instance of type <code>&lt;transform&gt;</code> .						
Description	<p>Creates a new transform by composing a given translation with the transform <i>transform</i>. The order of composition is <i>transform</i> first, followed by the translation transform.</p> <p>The argument <i>dx</i> represents the <i>delta</i> by which to translate the <i>x</i> coordinate.</p> <p>The argument <i>dy</i> represents the <i>delta</i> by which to translate the <i>y</i> coordinate.</p> <p>Note that this function could be implemented by using <code>make-translation-transform</code> and <code>compose-transforms</code>. It is provided because it is common to build up a transform as a series of simple transforms.</p>						
See also	See the functions <code>make-translation-transform</code> , page 39 and <code>compose-transforms</code> , page 21.						

	<b>compose-translation-with-transform</b>	<i>Generic function</i>
Summary	Creates a new transform by composing a transform with the given translation.	
Signature	<code>compose-translation-with-transform transform dx dy =&gt; transform</code>	
Arguments	<p><i>transform</i>      An instance of type <code>&lt;transform&gt;</code>.</p> <p><i>dx</i>                An instance of type <code>&lt;real&gt;</code>.</p> <p><i>dy</i>                An instance of type <code>&lt;real&gt;</code>.</p>	
Values	<i>transform</i> An instance of type <code>&lt;transform&gt;</code> .	
Description	<p>Creates a new transform by composing the transform <i>transform</i> with the given translation. The order of composition is that the translation transform is applied first, followed by the argument <i>transform</i>.</p> <p>The argument <i>dx</i> represents the <i>delta</i> by which to translate the <i>x</i> coordinate.</p> <p>The argument <i>dy</i> represents the <i>delta</i> by which to translate the <i>y</i> coordinate.</p> <p>Note that this function could be implemented by using <code>make-translation-transform</code> and <code>compose-transforms</code>. It is provided, because it is common to build up a transform as a series of simple transforms.</p>	
See also	See the functions <code>make-translation-transform</code> , page 39 and <code>compose-transforms</code> , page 21.	

	<b>do-coordinates</b>	<i>Function</i>
Summary	Applies a function to each coordinate pair in its argument list.	

Signature	<code>do-coordinates function coordinates =&gt; ()</code>	
Arguments	<i>function</i>	An instance of type <code>&lt;function&gt;</code> .
	<i>coordinates</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;real&gt;)</code> .
Values	None	
Description	Applies <i>function</i> to each coordinate pair in <i>coordinates</i> . The length of <i>coordinates</i> must be a multiple of 2. <i>Function</i> takes two arguments, the <i>x</i> and <i>y</i> value of each coordinate pair.	

## do-endpoint-coordinates *Function*

Summary	Applies a function to each coordinate pair in its argument list.	
Signature	<code>do-endpoint-coordinates function coordinates =&gt; ()</code>	
Arguments	<i>function</i>	An instance of type <code>&lt;function&gt;</code> .
	<i>coordinates</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;real&gt;)</code> .
Values	None	
Description	Applies <i>function</i> to each pair of coordinate pairs in <i>coordinates</i> . The arguments <i>coordinates</i> represents a set of line segments rather than a set of points: The length of this sequence must therefore be a multiple of 4. Function takes 4 arguments, $(x_1, y_1, x_2, y_2)$ .	

## do-regions *Generic function*

Summary	Calls a function on each region in a set of regions.
---------	--

Signature	<code>do-regions function region #key normalize? =&gt; ()</code>	
Arguments	<i>function</i>	An instance of type <code>&lt;function&gt;</code> .
	<i>region</i>	An instance of type <code>&lt;region&gt;</code> .
	<i>normalize?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#f</code> .
Values	None	
Description	<p>Calls <i>function</i> on each region in the region set <i>region</i>. This is often more efficient than calling <code>region-set-regions</code>. <i>function</i> is a function of one argument, a region. <i>Region</i> can be either a region set or a simple region, in which case <i>function</i> is called once on <i>region</i> itself. If <i>normalize</i> is supplied, it must be either <code>#"x-banding"</code> or <code>#"y-banding"</code>. If it is <code>#"x-banding"</code> and all the regions in <i>region</i> are axis-aligned rectangles, the result is normalized by merging adjacent rectangles with banding done in the x direction. If it is <code>#"y-banding"</code> and all the regions in <i>region</i> are rectangles, the result is normalized with banding done in the y direction. Normalizing a region set that is not composed entirely of axis-aligned rectangles using x- or y-banding causes DUIM to signal the <code>&lt;region-set-not-rectangular&gt;</code> error.</p>	

## even-scaling-transform? *Generic function*

Summary	Returns <code>#t</code> if the transform <i>transform</i> multiplies all x lengths and y lengths by the same magnitude, otherwise returns <code>#f</code> .	
Signature	<code>even-scaling-transform? transform =&gt; boolean</code>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
Values	<i>boolean</i>	An instance of type <code>&lt;boolean&gt;</code> .

Description	Returns <code>#t</code> if the transform <i>transform</i> multiplies all <i>x</i> lengths and <i>y</i> lengths by the same magnitude, otherwise returns <code>#f</code> . <code>even-scaling-transform?</code> includes pure reflections through vertical and horizontal lines.
-------------	---

## \$everywhere *Constant*

Summary	The region that includes all the points on the two-dimensional infinite drawing plane.
---------	--

Type	<code>&lt;region&gt;</code>
------	-----------------------------

Description	The region that includes all the points on the two-dimensional infinite drawing plane.
-------------	--

See also	<code>\$nowhere</code> , page 40
----------	----------------------------------

## fix-coordinate *Function*

Summary	Coerces the given coordinate into an <code>&lt;integer&gt;</code> .
---------	---

Signature	<code>fix-coordinate coordinate =&gt; integer</code>
-----------	--

Arguments	<code>coordinate</code>	An instance of type <code>&lt;real&gt;</code> .
-----------	-------------------------	---

Values	<code>integer</code>	An instance of type <code>&lt;integer&gt;</code> .
--------	----------------------	--

Description	Coerces the given coordinate into an <code>&lt;integer&gt;</code> .
-------------	---

## \$identity-transform *Constant*

Summary	An instance of a transform that is guaranteed to be an identity transform, that is, the transform that does nothing.
---------	--

Type	<code>&lt;transform&gt;</code>
Description	An instance of a transform that is guaranteed to be an identity transform, that is, the transform that does nothing.
See also	<code>identity-transform?</code> , page 29

## identity-transform? *Generic function*

Summary	Returns <code>#t</code> if a transform is equal (in the sense of <code>transform-equal</code> ) to the identity transform.
Signature	<code>identity-transform? transform =&gt; boolean</code>
Arguments	<code>transform</code> An instance of type <code>&lt;transform&gt;</code> .
Values	<code>boolean</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if the transform <code>transform</code> is equal (in the sense of <code>transform-equal</code> ) to the identity transform, otherwise returns <code>#f</code> .
See also	<code>\$identity-transform</code> , page 28

## invert-transform *Generic function*

Summary	Returns a transform that is the inverse of the given transform.
Signature	<code>invert-transform transform =&gt; transform</code>
Arguments	<code>transform</code> An instance of type <code>&lt;transform&gt;</code> .
Values	<code>transform</code> An instance of type <code>&lt;transform&gt;</code> .

Exceptions	If <i>transform</i> is singular, <code>invert-transform</code> signals the <code>&lt;singular-transform&gt;</code> error.
	<b>Note:</b> With finite-precision arithmetic there are several low-level conditions that might occur during the attempt to invert a singular or <i>almost</i> singular transform. (These include computation of a zero determinant, floating-point underflow during computation of the determinant, or floating-point overflow during subsequent multiplication.) <code>invert-transform</code> signals the <code>&lt;singular-transform&gt;</code> error for all of these cases.
Description	Returns a transform that is the inverse of the transform <i>transform</i> . The result of composing a transform with its inverse is equal to the identity transform.
See also	<code>invertible-transform?</code> , page 30

## **invertible-transform?** *Generic function*

Summary	Returns <code>#t</code> if the given transform has an inverse.	
Signature	<code>invertible-transform? transform =&gt; boolean</code>	
Arguments	<code>transform</code>	An instance of type <code>&lt;transform&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if the transform <i>transform</i> has an inverse, otherwise returns <code>#f</code> .	
See also	<code>invert-transform</code> , page 29	

## \$largest-coordinate *Constant*

Summary	The largest valid coordinate.
Type	<integer>
Description	The largest valid coordinate.
See also	<a href="#">\$smallest-coordinate</a> , page 56

## make-3-point-transform *Function*

Summary	Returns a transform that takes points <i>point-1</i> into <i>point-1-image</i> , <i>point-2</i> into <i>point-2-image</i> and <i>point-3</i> into <i>point-3-image</i> .
Signature	<pre>make-3-point-transform x1 y1 x2 y2 x3 y3 x1-image y1-image x2-image y2-image x3-image y3-image =&gt; transform</pre> <pre>make-3-point-transform* point-1 point-2 point-3 point-1-image point-2-image point-3-image =&gt; transform</pre>
Arguments	The following arguments are specific to <code>make-3-point-transform</code> . <ul style="list-style-type: none"> <li><i>x1</i> An instance of type &lt;real&gt;.</li> <li><i>y1</i> An instance of type &lt;real&gt;.</li> <li><i>x2</i> An instance of type &lt;real&gt;.</li> <li><i>y2</i> An instance of type &lt;real&gt;.</li> <li><i>x3</i> An instance of type &lt;real&gt;.</li> <li><i>y3</i> An instance of type &lt;real&gt;.</li> <li><i>x1-image</i> An instance of type &lt;real&gt;.</li> <li><i>y1-image</i> An instance of type &lt;real&gt;.</li> <li><i>x2-image</i> An instance of type &lt;real&gt;.</li> <li><i>y2-image</i> An instance of type &lt;real&gt;.</li> </ul>

	<i>x3-image</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>y3-image</i>	An instance of type <code>&lt;real&gt;</code> .
The following arguments are specific to <code>make-3-point-transform*</code> .		
	<i>point-1</i>	An instance of type <code>&lt;point&gt;</code> .
	<i>point-2</i>	An instance of type <code>&lt;point&gt;</code> .
	<i>point-3</i>	An instance of type <code>&lt;point&gt;</code> .
	<i>point-1-image</i>	An instance of type <code>&lt;point&gt;</code> .
	<i>point-2-image</i>	An instance of type <code>&lt;point&gt;</code> .
	<i>point-3-image</i>	An instance of type <code>&lt;point&gt;</code> .
Values	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
Exceptions	If <i>point-1</i> , <i>point-2</i> and <i>point-3</i> are colinear, the <code>&lt;transform-underspecified&gt;</code> error is signalled. If <i>point-1-image</i> , <i>point-2-image</i> and <i>point-3-image</i> are colinear, the resulting transform will be singular (that is, will have no inverse) but this is not an error.	
Description	<p>Returns a transform that takes points <i>point-1</i> into <i>point-1-image</i>, <i>point-2</i> into <i>point-2-image</i> and <i>point-3</i> into <i>point-3-image</i>. Three non-colinear points and their images under the transform are enough to specify any affine transformation.</p> <p>The function <code>make-3-point-transform*</code> is identical to <code>make-3-point-transform</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>	

**make-bounding-box***Function*

Summary	Returns an object of the class <code>&lt;bounding-box&gt;</code> .
---------	--

Signature	<code>make-bounding-box x1 y1 x2 y2 =&gt; box</code>	
Arguments	<code>x1</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>y1</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>x2</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>y2</code>	An instance of type <code>&lt;real&gt;</code> .
Values	<code>box</code>	An instance of type <code>&lt;bounding-box&gt;</code> .
Description	<p>Returns an object of the class <code>&lt;bounding-box&gt;</code> with the edges specified by <code>x1</code>, <code>y1</code>, <code>x2</code>, and <code>y2</code>. <code>x1</code>, <code>y1</code>, <code>x2</code>, and <code>y2</code> are canonicalized in the following way. The min point of the box has an <code>x</code> coordinate that is the smaller of <code>x1</code> and <code>x2</code> and a <code>y</code> coordinate that is the smaller of <code>y1</code> and <code>y2</code>. The max point of the box has an <code>x</code> coordinate that is the larger of <code>x1</code> and <code>x2</code> and a <code>y</code> coordinate that is the larger of <code>y1</code> and <code>y2</code>. (Therefore, in a right-handed coordinate system the canonicalized values of <code>x1</code>, <code>y1</code>, <code>x2</code>, and <code>y2</code> correspond to the left, top, right, and bottom edges of the box, respectively.)</p> <p>This is a convenient shorthand function for <code>make(&lt;bounding-box&gt;, left: top: right: bottom:)</code></p>	

		<i>Function</i>
Summary	Returns an object of class <code>&lt;point&gt;</code> .	
Signature	<code>make-point x y =&gt; point</code>	
Arguments	<code>x</code>	An instance of <code>&lt;real&gt;</code> .
	<code>y</code>	An instance of <code>&lt;real&gt;</code>
Values	<code>point</code>	An instance of type <code>&lt;point&gt;</code> .

Description	Returns an object of class <point> whose coordinates are <i>x</i> and <i>y</i> .
-------------	--

	<i>Function</i>												
<b>make-reflection-transform</b>													
Summary	Returns a transform that reflects every point through the line passing through the positions <i>x1,y1</i> and <i>x2,y2</i> or through the points <i>point1</i> and <i>point2</i> .												
Signature	<pre><b>make-reflection-transform</b> <i>x1 y1 x2 y2</i> =&gt; <i>transform</i></pre> <pre><b>make-reflection-transform*</b> <i>point1 point2</i> =&gt; <i>transform</i></pre>												
Arguments	<p>The following arguments are specific to <b>make-reflection-transform</b>.</p> <table> <tr> <td><i>x1</i></td><td>An instance of type &lt;real&gt;.</td></tr> <tr> <td><i>y1</i></td><td>An instance of type &lt;real&gt;.</td></tr> <tr> <td><i>x2</i></td><td>An instance of type &lt;real&gt;.</td></tr> <tr> <td><i>y2</i></td><td>An instance of type &lt;real&gt;.</td></tr> </table> <p>The following arguments are specific to <b>make-reflection-transform*</b>.</p> <table> <tr> <td><i>point1</i></td><td>An instance of type &lt;point&gt;. The first point.</td></tr> <tr> <td><i>point2</i></td><td>An instance of type &lt;point&gt;. The second point.</td></tr> </table>	<i>x1</i>	An instance of type <real>.	<i>y1</i>	An instance of type <real>.	<i>x2</i>	An instance of type <real>.	<i>y2</i>	An instance of type <real>.	<i>point1</i>	An instance of type <point>. The first point.	<i>point2</i>	An instance of type <point>. The second point.
<i>x1</i>	An instance of type <real>.												
<i>y1</i>	An instance of type <real>.												
<i>x2</i>	An instance of type <real>.												
<i>y2</i>	An instance of type <real>.												
<i>point1</i>	An instance of type <point>. The first point.												
<i>point2</i>	An instance of type <point>. The second point.												
Values	<table> <tr> <td><i>transform</i></td><td>An instance of type &lt;transform&gt;. The resultant transformation.</td></tr> </table>	<i>transform</i>	An instance of type <transform>. The resultant transformation.										
<i>transform</i>	An instance of type <transform>. The resultant transformation.												
Description	Returns a transform that reflects every point through the line passing through the positions <i>x1,y1</i> and <i>x2,y2</i> or through the points <i>point1</i> and <i>point2</i> .												

The arguments *x1* and *y1* represent the coordinates of the first point of reflection. The arguments *x2* and *y2* represent the coordinates of the second point of reflection.

A reflection is a transform that preserves lengths and magnitudes of angles, but changes the sign (or handedness) of angles. If you think of the drawing plane on a transparent sheet of paper, a reflection is a transformation that turns the paper over.

The function `make-reflection-transform*` is identical to `make-reflection-transform`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also	<code>make-rotation-transform</code> , page 35 <code>make-scaling-transform</code> , page 36 <code>make-transform</code> , page 38 <code>make-translation-transform</code> , page 39 <code>&lt;reflection-underspecified&gt;</code> , page 45
----------	---

	<i>Function</i>				
Summary	Returns a transform that rotates all points by <i>angle</i> around the point specified by coordinates <i>origin-x</i> and <i>origin-y</i> or the point object <i>origin</i> .				
Signature	<pre><code>make-rotation-transform angle #key origin-x origin-y =&gt; transform</code></pre> <pre><code>make-rotation-transform* angle #key origin =&gt; transform</code></pre>				
Arguments	<table> <tr> <td><i>angle</i></td> <td>An instance of type <code>&lt;real&gt;</code>.</td> </tr> <tr> <td colspan="2">The following arguments are specific to <code>make-rotation-transform</code>.</td> </tr> </table>	<i>angle</i>	An instance of type <code>&lt;real&gt;</code> .	The following arguments are specific to <code>make-rotation-transform</code> .	
<i>angle</i>	An instance of type <code>&lt;real&gt;</code> .				
The following arguments are specific to <code>make-rotation-transform</code> .					

	<i>origin-x</i>	An instance of type <code>&lt;real&gt;</code> . Default value: 0.
	<i>origin-y</i>	An instance of type <code>&lt;real&gt;</code> . Default value: 0.
The following argument is specific to <code>make-reflection-transform*</code> .		
	<i>origin</i>	An instance of type <code>&lt;point&gt;</code> . Default value: (0, 0).
Values	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
Description	<p>Returns a transform that rotates all points by <i>angle</i> around the point specified by coordinates <i>origin-x</i> and <i>origin-y</i> or the point object <i>origin</i>. The angle must be expressed in radians.</p> <p>A rotation is a transform that preserves length and angles of all geometric entities. Rotations also preserve one point (the origin) and the distance of all entities from that point.</p> <p>The function <code>make-rotation-transform*</code> is identical to <code>make-rotation-transform</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>	
See also	<p><code>make-reflection-transform</code>, page 34</p> <p><code>make-scaling-transform</code>, page 36</p> <p><code>make-transform</code>, page 38</p> <p><code>make-translation-transform</code>, page 39</p>	

	<b>make-scaling-transform</b>	<i>Function</i>
Summary	Returns a transform that multiplies the x-coordinate distance of every point from <i>origin</i> by <i>scale-x</i> and the y-coordinate distance of every point from <i>origin</i> by <i>scale-y</i> .	

Signature	<code>make-scaling-transform scale-x scale-y #key origin-x origin-y =&gt; transform</code>  <code>make-scaling-transform* scale-x scale-y #key origin =&gt; transform</code>	
Arguments	<code>scale-x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>scale-y</code>	An instance of type <code>&lt;real&gt;</code> .
	The following arguments are specific to <code>make-scaling-transform</code> .	
	<code>origin-x</code>	An instance of type <code>&lt;real&gt;</code> . Default value: 0.
	<code>origin-y</code>	An instance of type <code>&lt;real&gt;</code> . Default value: 0.
	The following argument is specific to <code>make-scaling-transform*</code> .	
	<code>origin</code>	An instance of type <code>&lt;point&gt;</code> .
Values	<code>transform</code>	An instance of type <code>&lt;transform&gt;</code> . The resultant transformation.
Description	<p>Returns a transform that multiplies the <i>x</i>-coordinate distance of every point from <i>origin</i> by <i>scale-x</i> and the <i>y</i>-coordinate distance of every point from <i>origin</i> by <i>scale-y</i>.</p> <p>The argument <i>scale-x</i> represents the scaling factor for the <i>x</i> direction.</p> <p>The argument <i>scale-y</i> represents the scaling factor for the <i>y</i> direction.</p> <p>The arguments <i>origin-x</i> and <i>origin-y</i> represent the point around which scaling is performed. The default is to scale around the origin.</p> <p>There is no single definition of a scaling transformation. Transforms that preserve all angles and multiply all lengths by the same factor (preserving the <i>shape</i> of all entities) are certainly scaling transformations. However, scaling is also used</p>	

to refer to transforms that scale distances in the  $x$  direction by one amount and distances in the  $y$  direction by another amount.

The function `make-scaling-transform*` is identical to `make-scaling-transform`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also [make-reflection-transform](#), page 34  
[make-rotation-transform](#), page 35  
[make-transform](#), page 38  
[make-translation-transform](#), page 39

		<i>Function</i>
Summary	Returns a general affine transform.	
Signature		<code>make-transform mxx mxy myx myy tx ty =&gt; transform</code>
Arguments	<code>mxx</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>mxy</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>myx</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>myy</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>tx</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>ty</code>	An instance of type <code>&lt;real&gt;</code> .
Values	<code>transform</code>	An instance of type <code>&lt;transform&gt;</code> .
Description	Returns a general transform whose effect is:	

$$\begin{aligned}x' &= m_{xx}x + m_{xy}y + t_x \\y' &= m_{yx}x + m_{yy}y + t_y\end{aligned}$$

where  $x$  and  $y$  are the coordinates of a point before the transform and  $x'$  and  $y'$  are the coordinates of the corresponding point after.

All of the arguments to `make-transform` must be real numbers.

This is a convenient shorthand for `make(<transform> ...)`.

See also	<code>make-reflection-transform</code> , page 34 <code>make-rotation-transform</code> , page 35 <code>make-scaling-transform</code> , page 36 <code>make-translation-transform</code> , page 39
----------	--

## make-translation-transform *Function*

**Summary** Returns a transform that translates all points by  $dx$  in the  $x$  direction and  $dy$  in the  $y$  direction.

**Signature** `make-translation-transform dx dy => transform`

**Arguments** `dx` An instance of type `<real>`.

`dy` An instance of type `<real>`.

**Values** `transform` An instance of type `<transform>`.

**Description** Returns a transform that translates all points by  $dx$  in the  $x$  direction and  $dy$  in the  $y$  direction.

The argument  $dx$  represents the *delta* by which to translate the  $x$  coordinate.

The argument  $dy$  represents the *delta* by which to translate the  $y$  coordinate.

A translation is a transform that preserves length, angle, and orientation of all geometric entities.

See also [make-reflection-transform](#), page 34  
[make-rotation-transform](#), page 35  
[make-scaling-transform](#), page 36  
[make-transform](#), page 38

## \$nowhere *Constant*

Summary The empty region, the opposite of \$everywhere.  
 Type `<region>`  
 Description The empty region, the opposite of \$everywhere.  
 See also [\\$everywhere](#), page 28

## <path> *Open abstract class*

Summary The class <path> denotes bounded regions that have dimensionality 1 (that is, have length).  
 Superclasses `<region>`  
 Init-keywords None.  
 Description The class <path> denotes bounded regions that have dimensionality 1 (that is, have length).  
 <path> is a subclass of <region>. Constructing a <path> object with no length (via `make-line*`, for example) may canonicalize it to \$nowhere.

Operations      The following operation is exported from the **DUIM-Geometry** module.

**path?**

See also      **path?**, page 41

## **path?** *Generic function*

Summary      Returns #t if its argument is a path.

Signature      **path? object => boolean**

Arguments      *object*      An instance of type <object>.

Values      *boolean*      An instance of type <boolean>.

Description      Returns #t if *object* is a path, otherwise returns #f.

See also      <path>, page 40

## <point> *Open abstract instantiable class*

Summary      The class that corresponds to a mathematical point.

Superclasses      <region>

Init-keywords      **x:**      An instance of type <integer>.

**y:**      An instance of type <integer>.

Description      The class that corresponds to a mathematical point. <point> is a subclass of <region>. The **x:** and **y:** init-keywords correspond to the x and y coordinates, respectively.

Operations	The following operations are exported from the <b>DUIM-Geometry</b> module.
	= box-edges point? point-position point-x point-y region-contains-position? region-contains-region? region-intersection region-intersects-region? transform-region

## point?

*Generic function*

Summary	Returns true if <i>object</i> is a point.	
Signature	<code>point? object =&gt; boolean</code>	
Arguments	<i>object</i>	An instance of type <code>&lt;object&gt;</code> .
Values	<i>boolean</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns #t if <i>object</i> is a point.	

## point-position

*Generic function*

Summary	Returns both the <i>x</i> and <i>y</i> coordinates of a point.	
Signature	<code>point-position point =&gt; x y</code>	
Arguments	<i>point</i>	An instance of type <code>&lt;point&gt;</code> .
Values	<i>x</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>y</i>	An instance of type <code>&lt;real&gt;</code> .
Description	Returns both the <i>x</i> and <i>y</i> coordinates of the point <i>point</i> as two values.	
See also	<code>point-x</code> , page 43	

`point-x`, page 43

**point-x** *Generic function*

Summary	Returns the <i>x</i> coordinate of a point.
Signature	<code>point-x point =&gt; x</code>
Arguments	<i>point</i> An instance of type <code>&lt;point&gt;</code> .
Values	<i>x</i> An instance of type <code>&lt;real&gt;</code> .
Description	Returns the <i>x</i> coordinate of <i>point</i> .
See also	<code>point-position</code> , page 42 <code>point-y</code> , page 43

**point-y** *Generic function*

Summary	Returns the <i>y</i> coordinate of a point.
Signature	<code>point-y point =&gt; y</code>
Arguments	<i>point</i> An instance of type <code>&lt;point&gt;</code> .
Values	<i>y</i> An instance of type <code>&lt;real&gt;</code>
Description	Returns the <i>y</i> coordinate of <i>point</i> .
See also	<code>point-position</code> , page 42 <code>point-x</code> , page 43

## rectilinear-transform?

*Generic function*

Summary	Returns #t if a transform always transforms any axis-aligned rectangle into another axis-aligned rectangle.	
Signature	<code>rectilinear-transform? transform =&gt; boolean</code>	
Arguments	<code>transform</code>	An instance of type <transform>.
Values	<code>boolean</code> An instance of type <boolean>.	
Description	<p>Returns #t if the transform <i>transform</i> always transforms any axis-aligned rectangle into another axis-aligned rectangle, otherwise returns #f.</p> <p>This category includes scalings as a subset, and also includes 90 degree rotations.</p> <p>Rectilinear transforms are the most general category of transforms for which the bounding rectangle of a transformed object can be found by transforming the bounding rectangle of the original object.</p>	

## reflection-transform?

*Generic function*

Summary	Returns #t if the transform inverts the <i>handedness</i> of the coordinate system.	
Signature	<code>reflection-transform? transform =&gt; boolean</code>	
Arguments	<code>transform</code>	An instance of type <transform>.
Values	<code>boolean</code> An instance of type <boolean>.	
Description	Returns #t if the transform <i>transform</i> inverts the <i>handedness</i> of the coordinate system, otherwise returns #f.	

Note that this is a very inclusive category — transforms are considered reflections even if they distort, scale, or skew the coordinate system, as long as they invert the handedness.

## **<reflection-underspecified>** *Sealed concrete class*

Summary	The error that is signalled when <code>make-reflection-transform</code> is given two coincident points.
Superclasses	<code>&lt;transform-underspecified&gt;</code>
Init-keywords	<code>points:</code> Instances of type <code>&lt;point&gt;</code> .
Description	The error that is signalled when <code>make-reflection-transform</code> is given two coincident points. This condition handles the <code>points:</code> initarg, which is used to supply the points that are in error.
Operations	None.
See also	<code>make-reflection-transform</code> , page 34

## **<region>** *Open abstract class*

Summary	The class that corresponds to a set of points.
Superclasses	<code>&lt;object&gt;</code>
Init-keywords	None.
Description	The class that corresponds to a set of points. The <code>&lt;region&gt;</code> class includes both bounded and unbounded regions.

There is no `make` method for `<region>` because of the impossibility of a uniform way to specify the arguments to such a function.

Operations	The following operations are exported from the <code>DUIM-Geometry</code> module.  <code>= do-regions region? region-contains-position? region-contains-region? region-difference region-empty? region-equal region-intersection region-intersects-region? region-set-function region-set-regions region-union</code>
See also	<code>region?</code> , page 46

## **region?** *Generic function*

Summary	Returns <code>#t</code> if its argument is a region.	
Signature	<code>region? object =&gt; boolean</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if <i>object</i> is a region, otherwise returns <code>#f</code> .	
See also	<code>&lt;region&gt;</code> , page 45	

## **region-contains-position?** *Generic function*

Summary	Returns <code>#t</code> if the point at <i>x,y</i> is contained in the region.
Signature	<code>region-contains-position? region x y =&gt; boolean</code>

Arguments	<i>region</i>	An instance of type <region>.
	<i>x</i>	An instance of type <real>.
	<i>y</i>	An instance of type <real>.
Values	<i>boolean</i>	An instance of type <boolean>.
Description		Returns #t if the point at <i>x,y</i> is contained in the region <i>region</i> , otherwise returns #f. Since regions in DUIM are closed, this returns #t if the point at <i>x,y</i> is on the region's boundary.
See also		<code>region-contains-region?</code> , page 47

## **region-contains-region?** *Generic function*

Summary	Returns #t if all points in the second region are members of the first region.	
Signature	<code>region-contains-region? <i>region1</i> <i>region2</i> =&gt; boolean</code>	
Arguments	<i>region1</i>	An instance of type <region>.
	<i>region2</i>	An instance of type <region>.
Values	<i>boolean</i>	An instance of type <boolean>.
Description		Returns #t if all points in the region <i>region2</i> are members of the region <i>region1</i> , otherwise returns #f. <code>region-contains-position?</code> is a special case of <code>region-contains-region?</code> in which the region is the point <i>x,y</i> .
See also	<code>region-contains-position?</code> , page 46.	

	<i>Generic function</i>				
<b>region-difference</b>					
Summary	Returns a region that contains all points in the region <i>region1</i> that are not in the region <i>region2</i> (possibly plus additional boundary points to make the result closed).				
Signature	<b>region-difference</b> <i>region1</i> <i>region2</i> => <i>region</i>				
Arguments	<table> <tr> <td><i>region1</i></td><td>An instance of type &lt;<i>region</i>&gt;.</td></tr> <tr> <td><i>region2</i></td><td>An instance of type &lt;<i>region</i>&gt;.</td></tr> </table>	<i>region1</i>	An instance of type < <i>region</i> >.	<i>region2</i>	An instance of type < <i>region</i> >.
<i>region1</i>	An instance of type < <i>region</i> >.				
<i>region2</i>	An instance of type < <i>region</i> >.				
Values	<i>region</i> An instance of type < <i>region</i> >.				
Description	<p>Returns a region that contains all points in the region <i>region1</i> that are not in the region <i>region2</i> (possibly plus additional boundary points to make the result closed).</p> <p>The result of <b>region-difference</b> has the same dimensionality as <i>region1</i>, or is <b>\$nowhere</b>. For example, the difference of an area and a path produces the same area; the difference of a path and an area produces the path clipped to stay outside of the area.</p> <p><b>Note:</b> <b>region-difference</b> may return either a simple region or a region set.</p>				

	<i>Generic function</i>		
<b>region-empty?</b>			
Summary	Returns #t if the region is empty.		
Signature	<b>region-empty?</b> <i>region</i> => <i>boolean</i>		
Arguments	<table> <tr> <td><i>region</i></td><td>An instance of type &lt;<i>region</i>&gt;.</td></tr> </table>	<i>region</i>	An instance of type < <i>region</i> >.
<i>region</i>	An instance of type < <i>region</i> >.		
Values	<i>boolean</i> An instance of type < <i>boolean</i> >.		
Description	Returns #t if the region is empty, otherwise returns #f.		

**region-equal***Generic function*

Summary	Returns <code>#t</code> if the two regions <i>region1</i> and <i>region2</i> contain exactly the same set of points.	
Signature	<code>region-equal region1 region2 =&gt; boolean</code>	
Arguments	<i>region1</i>	An instance of type <code>&lt;region&gt;</code> .
	<i>region2</i>	An instance of type <code>&lt;region&gt;</code> .
Values	<i>boolean</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if the two regions <i>region1</i> and <i>region2</i> contain exactly the same set of points, otherwise returns <code>#f</code> . There is a method <code>on = on</code> <code>&lt;region&gt;</code> and <code>&lt;region&gt;</code> that calls <code>region-equal</code> .	

**region-intersection***Generic function*

Summary	Returns the intersection of two regions, as a region.	
Signature	<code>region-intersection region1 region2 =&gt; region</code>	
Arguments	<i>region1</i>	An instance of type <code>&lt;region&gt;</code> .
	<i>region2</i>	An instance of type <code>&lt;region&gt;</code> .
Values	<i>region</i>	An instance of type <code>&lt;region&gt;</code> .
Description	Returns a region that contains all points that are in both of the regions <i>region1</i> and <i>region2</i> (possibly with some points removed in order to satisfy the dimensionality rule).  The result of <code>region-intersection</code> has dimensionality that is the minimum dimensionality of <i>region1</i> and <i>region2</i> , or is <code>\$nowhere</code> . For example, the intersection of two areas is either another area or <code>\$nowhere</code> ; the intersection of two paths is	

either another path or `$nowhere`; the intersection of a path and an area produces the path clipped to stay inside of the area.

**Note:** `region-intersection` may return either a simple region or a region set.

See also      [region-union](#), page 53

## **region-intersects-region?** *Generic function*

Summary	Returns <code>#f</code> if two regions do not intersect.	
Signature	<code>region-intersects-region? region1 region2 =&gt; boolean</code>	
Arguments	<code>region1</code>	An instance of type <code>&lt;region&gt;</code> .
	<code>region2</code>	An instance of type <code>&lt;region&gt;</code> .
Values	<code>boolean</code> An instance of type <code>&lt;boolean&gt;</code> .	
Description	Returns <code>#f</code> if <code>region-intersection</code> of the two regions <code>region1</code> and <code>region2</code> would be <code>\$nowhere</code> (that is, they do not intersect), otherwise returns <code>#t</code> .	

## **<region-set>** *Open abstract class*

Summary	The class that represents a region set.	
Superclasses	<code>&lt;region&gt;</code>	
Init-keywords	None.	
Description	The class that represents a region set; a subclass of <code>&lt;region&gt;</code> .	

Operations	The following operations are exported from the <code>DUIM-Geometry</code> module.
	<code>box-edges do-regions region-contains-position?</code> <code>region-contains-region? region-difference region-</code> <code>empty? region-intersection region-set-function</code> <code>region-set-regions region-union transform-region</code>

See also      `region-set?`, page 51

## **region-set?** *Generic function*

Summary	Returns <code>#t</code> if its argument is a region set.
Signature	<code>region-set? object =&gt; boolean</code>
Arguments	<code>object</code> An instance of type <code>&lt;object&gt;</code> .
Values	<code>boolean</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if <code>object</code> is a region set, otherwise returns <code>#f</code> .
See also	<code>&lt;region-set&gt;</code> , page 50

## **region-set-function** *Generic function*

Summary	Returns the function that composed the region.
Signature	<code>region-set-function region =&gt; function</code>
Arguments	<code>region</code> An instance of type <code>&lt;region&gt;</code> .
Values	<code>function</code> An instance of type <code>&lt;function&gt;</code> .

Description	Returns the function that composed the region, <code>intersection</code> , <code>union</code> , OR <code>difference</code> .
-------------	---

## **region-set-regions** *Generic function*

Summary	Returns a sequence of the regions in the region set.				
Signature	<code>region-set-regions region #key normalize? =&gt; regions</code>				
Arguments	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"><code>region</code></td> <td>An instance of type <code>&lt;region&gt;</code>.</td> </tr> <tr> <td><code>normalize?</code></td> <td><code>one-of(#f, #'x-banding", "y-banding")</code>. Default value: <code>#f</code>.</td> </tr> </table>	<code>region</code>	An instance of type <code>&lt;region&gt;</code> .	<code>normalize?</code>	<code>one-of(#f, #'x-banding", "y-banding")</code> . Default value: <code>#f</code> .
<code>region</code>	An instance of type <code>&lt;region&gt;</code> .				
<code>normalize?</code>	<code>one-of(#f, #'x-banding", "y-banding")</code> . Default value: <code>#f</code> .				
Values	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;"><code>regions</code></td> <td>An instance of type <code>limited(&lt;sequence&gt;, of: &lt;region&gt;)</code>.</td> </tr> </table>	<code>regions</code>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;region&gt;)</code> .		
<code>regions</code>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;region&gt;)</code> .				
Exceptions	Normalizing a region set that is not composed entirely of axis-aligned rectangles using x- or y-banding causes DUIM to signal the <code>&lt;region-set-not-rectangular&gt;</code> error.				
Description	<p>Returns a sequence of the regions in the region set <code>region</code>. <code>region</code> can be either a region set or a simple region, in which case the result is simply a sequence of one element: <code>region</code>.</p> <p>For the case of region sets that are unions of axis-aligned rectangles, the rectangles returned by <code>region-set-regions</code> are guaranteed not to overlap. If <code>normalize</code> is supplied, it must be either <code>#"x-banding"</code> OR <code>#"y-banding"</code>. If it is <code>#"x-banding"</code> and all the regions in <code>region</code> are axis-aligned rectangles, the result is normalized by merging adjacent rectangles with banding done in the x direction. If it is <code>#"y-banding"</code> and all the regions in <code>region</code> are rectangles, the result is normalized with banding done in the y direction.</p>				

## region-union *Generic function*

Summary	Returns the union of two regions, as a region.	
Signature	<code>region-union region1 region2 =&gt; region</code>	
Arguments	<i>region1</i>	An instance of type <code>&lt;region&gt;</code> .
	<i>region2</i>	An instance of type <code>&lt;region&gt;</code> .
Values	<i>region</i>	An instance of type <code>&lt;region&gt;</code> .
Description	<p>Returns a region that contains all points that are in either of the regions <i>region1</i> or <i>region2</i> (possibly with some points removed in order to satisfy the dimensionality rule)</p> <p>The result of <code>region-union</code> always has dimensionality that is the maximum dimensionality of <i>region1</i> and <i>region2</i>. For example, the union of a path and an area produces an area; the union of two paths is a path.</p> <p><b>Note:</b> <code>region-union</code> may return either a simple region or a region set.</p>	
See also	<code>region-intersection</code> , page 49	

## rigid-transform? *Generic function*

Summary	Returns <code>#t</code> if the <i>transform</i> transforms the coordinate system as a rigid object.	
Signature	<code>rigid-transform? transform =&gt; boolean</code>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
Values	<i>boolean</i>	An instance of type <code>&lt;boolean&gt;</code> .

Description	Returns <code>#t</code> if the <i>transform</i> transforms the coordinate system as a rigid object, that is, as a combination of translations, rotations, and pure reflections. Otherwise, it returns <code>#f</code> .  Rigid transforms are the most general category of transforms that preserve magnitudes of all lengths and angles.
-------------	---

## scaling-transform?

*Generic function*

Summary	Returns <code>#t</code> if the transform <i>transform</i> multiplies all <i>x</i> lengths by one magnitude and all <i>y</i> lengths by another magnitude, otherwise returns <code>#f</code> .
Signature	<code>scaling-transform? transform =&gt; boolean</code>
Arguments	<i>transform</i> An instance of type <code>&lt;transform&gt;</code> .
Values	<i>boolean</i> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if the transform <i>transform</i> multiplies all <i>x</i> lengths by one magnitude and all <i>y</i> lengths by another magnitude, otherwise returns <code>#f</code> . This category includes even scalings as a subset.

## set-box-edges

*Generic function*

Summary	Sets the edges of a box and returns the bounding box.
Signature	<code>set-box-edges box left top right bottom =&gt; box</code>
Arguments	<i>box</i> An instance of type <code>&lt;bounding-box&gt;</code> . <i>left</i> An instance of type <code>&lt;integer&gt;</code> . <i>top</i> An instance of type <code>&lt;integer&gt;</code> . <i>right</i> An instance of type <code>&lt;integer&gt;</code> .

	<i>bottom</i>	An instance of type <integer>.
Values	<i>box</i>	An instance of type <bounding-box>.
Description		Sets the edges of a box and returns the bounding box <i>box</i> . This might destructively modify <i>box</i> or it might not, depending on what class <i>box</i> is.

## **set-box-position** *Generic function*

Summary	Sets the position of the bounding box and returns a (possibly new) box.	
Signature	<b>set-box-position</b> <i>box</i> <i>x</i> <i>y</i> => <i>box</i>	
Arguments	<i>box</i>	An instance of type <bounding-box>.
	<i>x</i>	An instance of type <real>.
	<i>y</i>	An instance of type <real>.
Values	<i>box</i>	An instance of type <bounding-box>.
Description	Sets the position of the bounding box <i>box</i> and might or might not modify the box.	

## **set-box-size** *Generic function*

Summary	Sets the size (width and height) of the bounding box <i>box</i> .	
Signature	<b>set-box-size</b> <i>box</i> <i>width</i> <i>height</i> => <i>box</i>	
Arguments	<i>box</i>	An instance of type <bounding-box>.
	<i>width</i>	An instance of type <integer>.
	<i>height</i>	An instance of type <integer>

Values      *box*      An instance of type <bounding-box>.

Description      Sets the size (width and height) of the bounding box *box*.

## <singular-transform>                                  *Sealed instantiable class*

Summary      The error that is signalled when `invert-transform` is called on a singular transform, that is, a transform that has no inverse.

Superclasses      <transform-error>

Init-keywords      `transform:`      Used to supply the transform that is singular.

Description      The error that is signalled when `invert-transform` is called on a singular transform, that is, a transform that has no inverse.

This condition handles the `transform:` initarg, which is used to supply the transform that is singular.

Operations      None.

See also      `invert-transform`, page 29

## \$smallest-coordinate                                  *Constant*

Summary      The smallest valid coordinate.

Type      <integer>

Description      The smallest valid coordinate. Coordinates must be instances of type <integer>.

See also      `$largest-coordinate`, page 31

<b>&lt;transform&gt;</b>		<i>Open abstract instantiable class</i>
Summary		The superclass of all transforms.
Superclasses		<code>&lt;object&gt;</code>
Init-keywords	<code>mxx:</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>mxy:</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>myx:</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>myy:</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>tx:</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>ty:</code>	An instance of type <code>&lt;real&gt;</code> .
Description		The superclass of all transforms. There are one or more subclasses of <code>&lt;transform&gt;</code> with implementation-dependent names that implement transforms. The exact names of these classes is explicitly unspecified.  All of the instantiable transformation classes provided by DUIM are immutable.
Operations		The following operations are exported from the <code>DUIM-Geometry</code> module.  <code>= compose-rotation-with-transform compose-scaling-with-transform compose-transforms compose-transform-with-translation compose-translation-with-transform even-scaling-transform? identity-transform? invert-transform invertible-transform?</code> <code>rectilinear-transform? reflection-transform? rigid-transform? scaling-transform? transform? transform-angles transform-box transform-distance transform-position transform-region translation-transform?</code>

```
untransform-angles untransform-box
untransform-distance untransform-position
untransform-region
```

See also      [transform?](#), page 58

## **transform?** *Generic function*

Summary	Returns #t if its argument is a transform.	
Signature	<b>transform? object =&gt; boolean</b>	
Arguments	<i>object</i>	An instance of type <object>.
Values	<i>boolean</i>	An instance of type <boolean>.
Description	Returns #t if <i>object</i> is a transform, otherwise returns #f.	
See also	<a href="#">&lt;transform&gt;</a> , page 57	

## **transform-angles** *Generic function*

Summary	Applies the transform to the start and end angles of an object, and returns the transformed angles.	
Signature	<b>transform-angles transform start-angle end-angle =&gt; new-start new-end</b>	
Arguments	<i>transform</i>	An instance of type <transform>.
	<i>start-angle</i>	An instance of type <real>.
	<i>end-angle</i>	An instance of type <real>.
Values	<i>new-start</i>	An instance of type <real>.
	<i>new-end</i>	An instance of type <real>.

Description	Applies the transform <i>transform</i> to the angles <i>start-angle</i> and <i>end-angle</i> of an object, and returns the transformed angles.
-------------	--

## **transform-box** Generic function

Summary      Applies the transform to the rectangle specified by the four coordinate arguments.

Signature     **transform-box** *transform* *x1* *y1* *x2* *y2* => *left* *top* *right* *bottom*

Arguments    *transform*      An instance of type <**transform**>.

*x1*        An instance of type <**real**>.

*y1*        An instance of type <**real**>.

*x2*        An instance of type <**real**>.

*y2*        An instance of type <**real**>.

Values        *left*       An instance of type <**real**>.

*top*       An instance of type <**real**>.

*right*      An instance of type <**real**>.

*bottom*    An instance of type <**real**>.

Description    Applies the transform *transform* to the rectangle specified by the four coordinate arguments. **transform-box** is the spread version of **transform-region** in the case where the transform is rectilinear and the region is a rectangle.

The arguments *x1*, *y1*, *x2*, and *y2* are canonicalized and the four return values specify the minimum and maximum points of the transformed rectangle in the order *left*, *top*, *right*, and *bottom*.

An error is signalled if *transform* does not satisfy **rectilinear-transform?**.

	<i>Generic function</i>						
Summary	Applies a transform to a distance represented by the coordinate arguments and returns the transformed coordinates.						
Signature	<code>transform-distance transform dx dy =&gt; dx dy</code>						
Arguments	<table border="0"> <tr> <td><i>transform</i></td><td>An instance of type <code>&lt;transform&gt;</code>.</td></tr> <tr> <td><i>dx</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> <tr> <td><i>dy</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> </table>	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .	<i>dx</i>	An instance of type <code>&lt;real&gt;</code> .	<i>dy</i>	An instance of type <code>&lt;real&gt;</code> .
<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .						
<i>dx</i>	An instance of type <code>&lt;real&gt;</code> .						
<i>dy</i>	An instance of type <code>&lt;real&gt;</code> .						
Values	<table border="0"> <tr> <td><i>dx</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> <tr> <td><i>dy</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> </table>	<i>dx</i>	An instance of type <code>&lt;real&gt;</code> .	<i>dy</i>	An instance of type <code>&lt;real&gt;</code> .		
<i>dx</i>	An instance of type <code>&lt;real&gt;</code> .						
<i>dy</i>	An instance of type <code>&lt;real&gt;</code> .						
Description	Applies the transform <i>transform</i> to the distance represented by <i>dx</i> and <i>dy</i> , and returns the transformed <i>dx</i> and <i>dy</i> . A distance represents the difference between two points. It does not transform like a point.						
	<i>Sealed class</i>						
Summary	The superclass of all error conditions distributed when there is an error with a transform.						
Superclasses	<code>&lt;error&gt;</code>						
Init-keywords	None.						
Description	The class that is the superclass of three error conditions, <code>&lt;transform-underspecified&gt;</code> , <code>&lt;reflection-underspecified&gt;</code> , and <code>&lt;singular-transform&gt;</code> .						
Operations	None.						

## transform-position *Generic function*

Summary	Applies a transform to the point whose coordinates are <i>x</i> and <i>y</i> .	
Signature	<code>transform-position transform x y =&gt; new-x new-y</code>	
Arguments	<i>transform</i>	An instance of type < <code>transform</code> >.
	<i>x</i>	An instance of type < <code>real</code> >
	<i>y</i>	An instance of type < <code>real</code> >
Values	<i>new-x</i>	An instance of type < <code>real</code> >
	<i>new-y</i>	An instance of type < <code>real</code> >
Description	Applies the transform <i>transform</i> } to the point whose coordinates are <i>x</i> and <i>y</i> . <code>transform-position</code> is the <i>spread</i> version of <code>transform-region</code> in the case where the region is a point.	

## transform-region *Generic function*

Summary	Applies a transform to a region, and returns the transformed region.	
Signature	<code>transform-region transform region =&gt; region</code>	
Arguments	<i>transform</i>	An instance of type < <code>transform</code> >.
	<i>region</i>	An instance of type < <code>region</code> >.
Values	<i>region</i>	An instance of type < <code>region</code> >.
Description	Applies <i>transform</i> to the region <i>region</i> , and returns the transformed region.	

## <transform-underspecified> *Sealed concrete class*

Summary	The error that is signalled when <code>make-3-point-transform</code> is given three colinear image points.	
Superclasses	<transform-error>	
Init-keywords	<code>points:</code>	The points that are in error.
Description	The error that is signalled when <code>make-3-point-transform</code> is given three colinear image points. This condition handles the <code>points:</code> initarg, which is used to supply the points that are in error.	
Operations	None.	
See also	<code>make-3-point-transform</code> , page 31	

## translation-transform? *Generic function*

Summary	Returns #t if a transform is a pure translation, that is, a transform such that there are two distance components <code>transform dx</code> and <code>dy</code> and every point $(x,y)$ is moved to $(x+dx,y+dy)$ .	
Signature	<code>translation-transform? transform =&gt; boolean</code>	
Arguments	<code>transform</code>	An instance of type <transform>.
Values	<code>boolean</code>	An instance of type <boolean>.
Description	Returns #t if the transform <code>transform</code> is a pure translation, that is, a transform such that there are two distance components <code>transform dx</code> and <code>dy</code> and every point $(x,y)$ is moved to $(x+dx,y+dy)$ . Otherwise, <code>translation-transform?</code> returns #f.	

## untransform-angles Generic function

Summary	Undoes a transform and returns the original start and end angles of the object.	
Signature	<code>untransform-angles transform start-angle end-angle =&gt; orig-start orig-end</code>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
	<i>start-angle</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>end-angle</i>	An instance of type <code>&lt;real&gt;</code> .
Values	<i>orig-start</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>orig-end</i>	An instance of type <code>&lt;real&gt;</code> .
Exceptions	<code>&lt;singular-transform&gt;</code> cannot be inverted.	
Description	Undoes the transform <i>transform</i> to the angles <i>new-start</i> and <i>new-end</i> , returning the original <i>orig-start</i> and <i>orig-end</i> . This is exactly equivalent to:	
	<code>transform-angles(invert-transform(transform))</code>	

## untransform-box Generic function

Summary	Undoes the previous transformation on the rectangle <i>left</i> , <i>top</i> and <i>right</i> , <i>bottom</i> , returning the original box.	
Signature	<code>untransform-box transform x1 y1 x2 y2 =&gt; left top right bottom</code>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
	<i>x1</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>y1</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>x2</i>	An instance of type <code>&lt;real&gt;</code> .

	<i>y2</i>	An instance of type <real>.
Values	<i>left</i>	An instance of type <real>.
	<i>top</i>	An instance of type <real>.
	<i>right</i>	An instance of type <real>.
	<i>bottom</i>	An instance of type <real>.
Exceptions	<singular-transform> cannot be inverted.	
Description	Undoes the previous transformation on the rectangle <i>top-left-s</i> , <i>top-left-y</i> and <i>bottom-right-x</i> , <i>bottom-right-y</i> , returning the original box. This is exactly equivalent to:	
	<code>transform-box(invert-transform(<i>transform</i>))</code>	

		<i>Generic function</i>
Summary	Undoes the previous transformation on the distance <i>dx,dy</i> , returning the original <i>dx,dy</i> .	
Signature	<code>untransform-distance <i>transform dx dy =&gt; dx dy</i></code>	
Arguments	<i>transform</i>	An instance of type <transform>.
	<i>dx</i>	An instance of type <real>.
	<i>dy</i>	An instance of type <real>.
Values	<i>dx</i>	An instance of type <real>.
	<i>dy</i>	An instance of type <real>.
Exceptions	<singular-transform> cannot be inverted.	
Description	Undoes the previous transformation on the distance <i>dx,dy</i> , returning the original <i>dx,dy</i> . This is exactly equivalent to:	
	<code>transform-position(invert-transform(<i>transform</i>))</code>	

**untransform-position***Generic function*

Summary	Undoes the previous transformation on the point <i>x,y</i> , returning the original point.	
Signature	<code>untransform-position transform x y =&gt; x y</code>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
	<i>x</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>y</i>	An instance of type <code>&lt;real&gt;</code> .
Values	<i>x</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>y</i>	An instance of type <code>&lt;real&gt;</code> .
Exceptions	<code>&lt;singular-transform&gt;</code> cannot be inverted.	
Description	Undoes the previous transformation on the point <i>x,y</i> , returning the original point. This is exactly equivalent to:  <code>transform-position(invert-transform(transform))</code>	

**untransform-region***Generic function*

Summary	Undoes the previous transformation on a region, returning the original region.	
Signature	<code>untransform-region transform region2 =&gt; region1</code>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
	<i>region2</i>	An instance of type <code>&lt;region&gt;</code> . The region to untransform.
Values	<i>region1</i>	An instance of type <code>&lt;region&gt;</code> . The original region.

Exceptions	< <b>singular-transform</b> > cannot be inverted.
Description	Undoes the previous transformation on the region <i>region</i> , returning the original region. This is exactly equivalent to <code>transform-region(invert-transform(<b>transform region</b>))</code>

# 3

---

---

# DUIM-Extended-Geometry Library

## 3.1 Overview

The DUIM-Extended-Geometry library builds on the features provided by the DUIM-Geometry library, and provides more extensive support for coordinate geometry that is only required for more specialist uses. The library contains a single module, `duim-extended-geometry`, from which all the interfaces described in this chapter are exposed. Section 3.3 on page 68 contains complete reference entries for each exposed interface.

## 3.2 The class hierarchy for DUIM-Extended-Geometry

The DUIM-Extended-Geometry library defines no base classes itself, but instead subclasses two classes exposed in the DUIM-Geometry library: `<area>` and `<path>`. In each case, these subclasses provide more specialized geometrical tools.

### 3.2.1 Subclasses of `<area>`

Three subclasses of `<area>` are exposed in the DUIM-Extended-Geometry library, each of which provides the ability to create instances of particular shapes. Their usage is relatively obvious.

- <rectangle> This class is used to create rectangular shapes on a drawable object.
- <ellipse> This class is used to create elliptical shapes on a drawable object.
- <polygon> This class is used to create more general polygon shapes on a drawable object.

### 3.2.2 Subclass of <path>

Three subclasses of <path> are exposed in the DUIM-Extended-Geometry library, each of which provides the ability to create instances of particular types of line. Their usage is relatively obvious.

- <line> This class is used to create straight lines between two points on a drawable object.
- <elliptical-arc> This class is used to create elliptical arcs (portions of an ellipse) on a drawable object.
- <polyline> This class is used to create lines that pass through an arbitrary set of coordinates. It produces a jagged line with vertices at each coordinate.

## 3.3 DUIM-Extended-Geometry Module

This section contains a complete reference of all the interfaces that are exported from the `duim-extended-geometry` module.

		<i>Generic function</i>
do-polygon-coordinates		
Summary	Applies a function to all of the coordinates of the vertices of a polygon.	
Signature	<code>do-polygon-coordinates <i>function</i> <i>polygon</i> =&gt; ()</code>	
Arguments	<i>function</i>	An instance of type < <i>function</i> >.

*polygon* An instance of type <**polygon**>.

Values None.

Description Applies *function* to all of the coordinates of the vertices of *polygon*. *function* is a function of two arguments, the *x* and *y* coordinates of the vertex. **do-polygon-coordinates** returns #f.

See also **do-polygon-segments**, page 69

## do-polygon-segments *Generic function*

Summary Applies a function to the segments that compose a polygon.

Signature **do-polygon-segments** *function polygon => ()*

Arguments *function* An instance of type <**function**>. *polygon* An instance of type <**polygon**>.

Values None.

Description Applies *function* to the segments that compose *polygon*. *function* is a function of four arguments, the *x* and *y* coordinates of the start of the segment, and the *x* and *y* coordinates of the end of the segment. When **do-polygon-segments** is called on a closed polyline, it calls *function* on the segment that connects the last point back to the first point.

The function **do-polygon-segments** returns #f.

See also **do-polygon-coordinates**, page 68

<b>draw-design</b>	<i>Generic function</i>
Summary	Draws a design on a drawing surface.
Signature	<code>draw-design <i>drawable</i> <i>design</i> =&gt; ()</code>
Arguments	<p><i>drawable</i>      An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</p> <p><i>design</i>      A <code>&lt;region&gt;</code> to draw.</p>
Values	None.
Description	Draws <i>design</i> on the sheet medium <i>drawable</i> .
<b>&lt;ellipse&gt;</b>	<i>Abstract instantiable class</i>
Summary	The class that corresponds to an ellipse.
Superclasses	<code>&lt;area&gt;</code>
Init-keywords	<p><code>center-x:</code>      An instance of type <code>&lt;real&gt;</code>.</p> <p><code>center-y:</code>      An instance of type <code>&lt;real&gt;</code>.</p> <p><code>center-point:</code>      An instance of type <code>&lt;point&gt;</code>.</p> <p><code>radius-1-dx:</code>      An instance of type <code>&lt;real&gt;</code></p> <p><code>radius-1-dy:</code>      An instance of type <code>&lt;real&gt;</code></p> <p><code>radius-2-dx:</code>      An instance of type <code>&lt;real&gt;</code></p> <p><code>radius-2-dy:</code>      An instance of type <code>&lt;real&gt;</code></p> <p><code>start-angle:</code>      An instance of <code>false-or(&lt;real&gt;)</code>.</p> <p><code>end-angle:</code>      An instance of <code>false-or(&lt;real&gt;)</code>.</p>
Description	An <i>ellipse</i> is an area that is the outline and interior of an ellipse. Circles are special cases of ellipses.

The **center-x**: init-keyword specifies the **x** coordinate of the center of the ellipse.

The **center-y**: init-keyword specifies the **y** coordinate of the center of the ellipse.

The **center-point**: init-keyword specifies the center of the ellipse as a point.

An ellipse is specified in a manner that is easy to transform, and treats all ellipses on an equal basis. An ellipse is specified by its center point and two vectors that describe a bounding parallelogram of the ellipse. The bounding parallelogram is made by adding and subtracting the vectors from the center point in the following manner:

	<i>x</i> coordinate	<i>y</i> coordinate
Center of ellipse	$x_c$	$y_c$
Vectors	$dx_1$	$dy_1$
	$dx_2$	$dy_2$
Corners of parallelogram	$x_c + dx_1 + dx_2$	$y_c + dx_1 + dx_2$
	$x_c + dx_1 - dx_2$	$y_c + dx_1 - dx_2$
	$x_c - dx_1 - dx_2$	$y_c - dx_1 - dx_2$
	$x_c - dx_1 + dx_2$	$y_c - dx_1 + dx_2$

**Table 3.1**

Note that several different parallelograms specify the same ellipse. One parallelogram is bound to be a rectangle — the vectors will be perpendicular and correspond to the semi-axes of the ellipse.

## Operations

The following operations are exported from the **DUIM-Extended-Geometry** module.

```
draw-design ellipse? ellipse-center-point ellipse-
center-position ellipse-end-angle ellipse-radii
ellipse-start-angle
```

The following operations are exported from the **DUIM-Geometry** module.

```
box-edges transform-region
```

See also      [`<area>`](#), page 10

[`make-ellipse`](#), page 80

## **ellipse?** *Generic function*

Summary      Returns #t if an object is an ellipse.

Signature      **ellipse? object => boolean**

Arguments      *object*      An instance of type [`<object>`](#).

Values      *boolean*      An instance of type [`<boolean>`](#).

Description      Returns #t if *object* is an ellipse, otherwise returns #f.

See Also      [`<ellipse>`](#), page 70

## **ellipse-center-point** *Generic function*

Summary      Returns the center point of an ellipse or an elliptical arc.

Signature      **ellipse-center-point *elliptical-object* => point**

Arguments      *elliptical-object*      An instance of type [`type-union\(<ellipse>, <elliptical-arc>\)`](#).

Values	<i>point</i>	An instance of type < <b>point</b> >.
Description	Returns the center point of <i>ellipse-object</i> as a < <b>point</b> > object.	
See also	<b>make-ellipse</b> , page 80	

## ellipse-center-position *Generic function*

Summary	Returns the coordinates of the center point of an ellipse or an elliptical arc.
Signature	<b>ellipse-center-position</b> <i>elliptical-object</i> => <i>x y</i>
Arguments	<i>elliptical-object</i> An instance of type <b>type-union(&lt;ellipse&gt;, &lt;elliptical-arc&gt;)</b> .
Values	<i>x</i> An instance of type < <b>real</b> >. <i>y</i> An instance of type < <b>real</b> >.
Description	Returns the coordinates of the center point of <i>elliptical-object</i> . The arguments <i>x</i> and <i>y</i> represent the <i>x</i> and <i>y</i> coordinates of the center of the elliptical object, respectively.
See also	<b>make-ellipse</b> , page 80

## ellipse-end-angle *Generic function*

Summary	Returns the end angle of an ellipse or an elliptical-object.
Signature	<b>ellipse-end-angle</b> <i>elliptical-object</i> => <i>angle</i>
Arguments	<i>elliptical-object</i> An instance of type <b>type-union(&lt;ellipse&gt;, &lt;elliptical-arc&gt;)</b> .

Values	<i>angle</i>	An instance of type <b>false-or(&lt;real&gt;)</b> .
Description		Returns the end angle of <i>elliptical-object</i> . If <i>elliptical-object</i> is a full ellipse or closed path then <b>ellipse-end-angle</b> returns #f; otherwise the value is a number greater than zero, and less than or equal to 2p.
See also		<b>make-ellipse</b> , page 80

## **ellipse-radii** *Generic function*

Summary	Returns four values corresponding to the two radius vectors of an elliptical arc.	
Signature	<b>ellipse-radii</b> <i>elliptical-object</i> => <i>r1-dx</i> <i>r1-dy</i> <i>r2-dx</i> <i>d2-dy</i>	
Arguments	<i>elliptical-object</i>	An instance of type <b>type-union(&lt;ellipse&gt;, &lt;elliptical-arc&gt;)</b> .
Values	<i>r1-dx</i>	An instance of type <b>&lt;real&gt;</b> .
	<i>r1-dy</i>	An instance of type <b>&lt;real&gt;</b> .
	<i>r2-dx</i>	An instance of type <b>&lt;real&gt;</b> .
	<i>d2-dy</i>	An instance of type <b>&lt;real&gt;</b> .
Description	Returns four values corresponding to the two radius vectors of <i>elliptical-object</i> . These values may be canonicalized in some way, and so may not be the same as the values passed to the constructor function.	
See also	<b>make-ellipse</b> , page 80	

	<i>Generic function</i>
Summary	Returns the start angle of an elliptical arc.
Signature	<code>ellipse-start-angle <i>elliptical-object</i> =&gt; <i>angle</i></code>
Arguments	<i>elliptical-object</i> An instance of type <code>type-union(&lt;ellipse&gt;, &lt;elliptical-arc&gt;)</code> .
Values	<i>angle</i> An instance of type <code>false-or(&lt;real&gt;)</code> .
Description	Returns the start angle of <i>elliptical-object</i> . If <i>elliptical-object</i> is a full ellipse or closed path then <code>ellipse-start-angle</code> returns <code>#f</code> ; otherwise the value will be a number greater than or equal to zero, and less than <code>2p</code> .
See also	<code>make-ellipse</code> , page 80

	<i>Abstract instantiable class</i>
Summary	An <i>elliptical arc</i> is a path consisting of all or a portion of the outline of an ellipse.
Superclasses	<code>&lt;path&gt;</code>
Init-keywords	<ul style="list-style-type: none"> <li><code>center-x:</code> An instance of type <code>&lt;real&gt;</code>.</li> <li><code>center-y:</code> An instance of <code>&lt;real&gt;</code>.</li> <li><code>center-point:</code> An instance of type <code>&lt;point&gt;</code>.</li> <li><code>radius-1-dx:</code> An instance of <code>&lt;real&gt;</code>.</li> <li><code>radius-1-dy:</code> An instance of <code>&lt;real&gt;</code>.</li> <li><code>radius-2-dx:</code> An instance of <code>&lt;real&gt;</code>.</li> <li><code>radius-2-dy:</code> An instance of <code>&lt;real&gt;</code>.</li> <li><code>start-angle:</code> An instance of <code>false-or(&lt;real&gt;)</code>.</li> </ul>

	<b>end-angle:</b> An instance of <code>false-or(&lt;real&gt;)</code> .
Description	An <i>elliptical arc</i> is a path consisting of all or a portion of the outline of an ellipse. Circular arcs are special cases of elliptical arcs.
Operations	<p>The following operations are exported from the <code>DUIM-Extended-Geometry</code> module.</p> <pre>draw-design ellipse-center-point ellipse-center-position ellipse-end-angle ellipse-radii ellipse-start-angle elliptical-arc?</pre> <p>The following operations are exported from the <code>DUIM-Geometry</code> module.</p> <pre>box-edges transform-region</pre>
See also	<p><code>elliptical-arc?</code>, page 76</p> <p><code>make-elliptical-arc</code>, page 81</p>

<b>elliptical-arc?</b>		<i>Generic function</i>
Summary	Returns <code>#t</code> if an object is an elliptical arc,	
Signature	<code>elliptical-arc? object =&gt; boolean</code>	
Arguments	<i>object</i>	An instance of type <code>&lt;object&gt;</code> .
Values	<i>boolean</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if <i>object</i> is an elliptical arc, otherwise returns <code>#f</code> .	
See also	<code>&lt;elliptical-arc&gt;</code> , page 75	

<line>		<i>Abstract instantiable class</i>
Summary		The class that corresponds to a line.
Superclasses		<path>
Init-keywords	<b>start-x:</b>	An instance of <real>.
	<b>start-y:</b>	An instance of <real>.
	<b>end-x:</b>	An instance of <real>.
	<b>end-y:</b>	An instance of <real>.
	<b>points:</b>	Instances of <point>.
Description		<p>The class that corresponds to a line. This is a subclass of &lt;path&gt;.</p> <p>This is the instantiable class that implements a line segment. <code>make-line</code> instantiates an object of type &lt;line&gt;.</p>
Operations		<p>The following operations are exported from the <b>DUIM-Extended-Geometry</b> module.</p> <pre>do-polygon-coordinates do-polygon-segments draw- design line? line-end-point line-end-position line- start-point line-start-position polygon-coordinates polygon-points polyline-closed?</pre> <p>The following operations are exported from the <b>DUIM-Geometry</b> module.</p> <pre>box-edges transform-region</pre>
See also		<p>&lt;path&gt;, page 40</p> <p><code>make-line</code>, page 83</p>

**line?** *Generic function*

Summary	Returns #t if an object is a line.
Signature	<code>line? object =&gt; boolean</code>
Arguments	<code>object</code> An instance of type <object>.
Values	<code>boolean</code> An instance of type <boolean>.
Description	Returns #t if <i>object</i> is a line, otherwise returns #f.

**line-end-point** *Generic function*

Summary	Returns the ending point of a line.
Signature	<code>line-end-point line =&gt; point</code>
Arguments	<code>line</code> An instance of type <line>.
Values	<code>point</code> An instance of type <point>.
Description	Returns the ending point of <i>line</i> as a <point> object.
See also	<code>line-start-point</code> , page 79

**line-end-position** *Generic function*

Summary	Returns the ending point of a line.
Signature	<code>line-end-position line =&gt; xy</code>
Arguments	<code>line</code> An instance of type <line>.
Values	<code>x</code> An instance of type <real>.

<i>y</i>	An instance of type <real>.
Description	Returns two real numbers representing the <i>x</i> and <i>y</i> coordinates of the ending point of <i>line</i> .  The arguments <i>x</i> and <i>y</i> represent the <i>x</i> and <i>y</i> coordinates of the end of the line, respectively.
See also	<a href="#">line-start-position</a> , page 79

## line-start-point

*Generic function*

Summary	Returns the starting point of a line.	
Signature	<code>line-start-point <i>line</i> =&gt; <i>point</i></code>	
Arguments	<i>line</i>	An instance of type <line>.
Values	<i>point</i>	An instance of type <point>.
Description	Returns the starting point of <i>line</i> as a <point> object.	
See also	<a href="#">line-end-point</a> , page 78	

## line-start-position

*Generic function*

Summary	Returns the starting point of a line.	
Signature	<code>line-start-position <i>line</i> =&gt; <i>x</i> <i>y</i></code>	
Arguments	<i>line</i>	An instance of type <line>.
Values	<i>x</i>	An instance of type <real>.
	<i>y</i>	An instance of type <real>.

Description	Returns two real numbers representing the <i>x</i> and <i>y</i> coordinates of the starting point of <i>line</i> .  The arguments <i>x</i> and <i>y</i> represent the <i>x</i> and <i>y</i> coordinates of the start of the line, respectively.
See also	<code>line-end-position</code> , page 78

	<i>Function</i>																		
Summary	Returns an object of class <code>&lt;ellipse&gt;</code> .																		
Signature	<code>make-ellipse center-x center-y radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key start-angle end-angle =&gt; ellipse</code> <code>make-ellipse* center-point radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key start-angle end-angle =&gt; ellipse</code>																		
Arguments	<table> <tr> <td><i>radius-1-dx</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> <tr> <td><i>radius-1-dy</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> <tr> <td><i>radius-2-dx</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> <tr> <td><i>radius-2-dy</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> <tr> <td><i>start-angle</i></td><td>An instance of type <code>false-or(&lt;real&gt;)</code>.</td></tr> <tr> <td><i>end-angle</i></td><td>An instance of type <code>false-or(&lt;real&gt;)</code>.</td></tr> </table> <p>The following arguments are specific to <code>make-ellipse</code>.</p> <table> <tr> <td><i>center-x</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> <tr> <td><i>center-y</i></td><td>An instance of type <code>&lt;real&gt;</code>.</td></tr> </table> <p>The following argument is specific to <code>make-ellipse*</code>.</p> <table> <tr> <td><i>center-point</i></td><td>An instance of type <code>&lt;point&gt;</code>.</td></tr> </table>	<i>radius-1-dx</i>	An instance of type <code>&lt;real&gt;</code> .	<i>radius-1-dy</i>	An instance of type <code>&lt;real&gt;</code> .	<i>radius-2-dx</i>	An instance of type <code>&lt;real&gt;</code> .	<i>radius-2-dy</i>	An instance of type <code>&lt;real&gt;</code> .	<i>start-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .	<i>end-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .	<i>center-x</i>	An instance of type <code>&lt;real&gt;</code> .	<i>center-y</i>	An instance of type <code>&lt;real&gt;</code> .	<i>center-point</i>	An instance of type <code>&lt;point&gt;</code> .
<i>radius-1-dx</i>	An instance of type <code>&lt;real&gt;</code> .																		
<i>radius-1-dy</i>	An instance of type <code>&lt;real&gt;</code> .																		
<i>radius-2-dx</i>	An instance of type <code>&lt;real&gt;</code> .																		
<i>radius-2-dy</i>	An instance of type <code>&lt;real&gt;</code> .																		
<i>start-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .																		
<i>end-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .																		
<i>center-x</i>	An instance of type <code>&lt;real&gt;</code> .																		
<i>center-y</i>	An instance of type <code>&lt;real&gt;</code> .																		
<i>center-point</i>	An instance of type <code>&lt;point&gt;</code> .																		
Values	<i>ellipse</i> An instance of type <code>&lt;ellipse&gt;</code> .																		

Description	Returns an object of class <code>&lt;ellipse&gt;</code> . The center of the ellipse is at the position <code>center-x,center-y</code> or the point <code>center-point</code> .  Two vectors, $(radius-1-dx, radius-1-dy)$ and $(radius-2-dx, radius-2-dy)$ specify the bounding parallelogram of the ellipse. All of the radii are real numbers. If the two vectors are colinear, the ellipse is not well-defined and the <code>ellipse-not-well-defined</code> error is signalled. The special case of an ellipse with its axes aligned with the coordinate axes can be obtained by setting both <code>radius-1-dy</code> and <code>radius-2-dx</code> to 0.  If <code>start-angle</code> or <code>end-angle</code> are supplied, the ellipse is the <i>pie slice</i> area swept out by a line from the center of the ellipse to a point on the boundary as the boundary point moves from the angle <code>start-angle</code> to <code>end-angle</code> . Angles are measured counter-clockwise with respect to the positive <code>x</code> axis. If <code>end-angle</code> is supplied, the default for <code>start-angle</code> is <code>0</code> ; if <code>start-angle</code> is supplied, the default for <code>end-angle</code> is <code>2π</code> ; if neither is supplied then the region is a full ellipse and the angles are meaningless.
See also	The function <code>make-ellipse*</code> is identical to <code>make-ellipse</code> , except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also See the class `<ellipse>`, page 70.

	<i>Function</i>
Summary	Returns an object of class <code>&lt;elliptical-arc&gt;</code> .
Signature	<code>make-elliptical-arc center-x center-y radius-1-dx radius-1-dy      radius-2-dx radius-2-dy #key start-angle end-angle =&gt; arc</code>  <code>make-elliptical-arc* center-point radius-1-dx radius-1-dy      radius-2-dx radius-2-dy #key start-angle end-angle =&gt; arc</code>
Arguments	<code>radius-1-dx</code> An instance of type <code>&lt;real&gt;</code> .

<i>radius-1-dy</i>	An instance of type <code>&lt;real&gt;</code> .
<i>radius-2-dx</i>	An instance of type <code>&lt;real&gt;</code> .
<i>radius-2-dy</i>	An instance of type <code>&lt;real&gt;</code> .
<i>start-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .
<i>end-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .
The following arguments are specific to <code>make-elliptical-arc</code> .	
<i>center-x</i>	An instance of type <code>&lt;real&gt;</code> .
<i>center-y</i>	An instance of type <code>&lt;real&gt;</code> .
The following argument is specific to <code>make-elliptical-arc*</code> .	
<i>center-point</i>	An instance of type <code>&lt;point&gt;</code> .
Values	<i>arc</i> An instance of type <code>&lt;elliptical-arc&gt;</code> .
Description	<p>Returns an object of class <code>&lt;elliptical-arc&gt;</code>. The center of the ellipse is at the position <i>center-x</i>,<i>center-y</i> or the point <i>center-point</i></p> <p>Two vectors, (<i>radius-1-dx</i>,<i>radius-1-dy</i>) and (<i>radius-2-dx</i>,<i>radius-2-dy</i>), specify the bounding parallelogram of the ellipse. All of the radii are real numbers. If the two vectors are colinear, the ellipse is not well-defined and the <code>ellipse-not-well-defined</code> error will be signalled. The special case of an elliptical arc with its axes aligned with the coordinate axes can be obtained by setting both <i>radius-1-dy</i> and <i>radius-2-dx</i> to 0.</p> <p>If <i>start-angle</i> and <i>end-angle</i> are supplied, the arc is swept from <i>start-angle</i> to <i>end-angle</i>. Angles are measured counter-clockwise with respect to the positive x axis. If <i>end-angle</i> is supplied, the default for <i>start-angle</i> is 0; if <i>start-angle</i> is supplied, the default for <i>end-angle</i> is 2π; if neither is supplied then the region is a closed elliptical path and the angles are meaningless.</p>

The function `make-elliptical-arc*` is identical to `make-elliptical-arc`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also See the class `<elliptical-arc>`, page 75.

## **make-line** *Function*

Summary Returns an object of class `<line>.t`

Signature `make-line start-x start-y end-x end-y => line`  
`make-line* start-point end-point => line`

Arguments	<code>start-x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>start-y</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>end-x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>end-y</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>start-point</code>	An instance of type <code>&lt;point&gt;</code> .
	<code>end-point</code>	An instance of type <code>&lt;point&gt;</code> .

Values `line` An instance of type `<line>`.

Description Returns an object of class `<line>` that connects the two positions (`start-x,start-y`) and (`end-x,end-y`) or the two points `start-point` and `end-point`.

## **make-polygon** *Function*

Summary Returns an object of class `<polygon>`.

Signature	<code>make-polygon coord-seq =&gt; polygon</code>
	<code>make-polygon* point-seq =&gt; polygon</code>
Arguments	The following argument is specific to <code>make-polygon</code> .  <code>coord-seq</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;real&gt;)</code> .
	The following argument is specific to <code>make-polygon*</code> .  <code>point-seq</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;point&gt;)</code> .
Values	<code>polygon</code> An instance of type <code>&lt;polygon&gt;</code> .
Description	Returns an object of class <code>&lt;polygon&gt;</code> consisting of the area contained in the boundary that is specified by the segments connecting each of the points in <code>point-seq</code> or the points represented by the coordinate pairs in <code>coord-seq</code> . <code>point-seq</code> is a sequence of points; <code>coord-seq</code> is a sequence of coordinate pairs, which are real numbers. It is an error if <code>coord-seq</code> does not contain an even number of elements.  The function <code>make-polygon*</code> is identical to <code>make-polygon</code> , except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

	<i>Function</i>
Summary	Returns an object of class <code>&lt;polyline&gt;</code> .
Signature	<code>make-polyline coord-seq #key closed? =&gt; polyline</code>
	<code>make-polyline* point-seq #key closed? =&gt; polyline</code>
Arguments	<code>closed?</code> An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#f</code> .

The following argument is specific to `make-polyline`.

*coord-seq* An instance of type `limited(<sequence>, of: <real>)`.

The following argument is specific to `make-polyline*`.

*point-seq* An instance of type `limited(<sequence>, of: <point>)`.

Values *polyline* An instance of type `<polyline>`.

Description Returns an object of class `<polyline>` consisting of the segments connecting each of the points in *point-seq* or the points represented by the coordinate pairs in *coord-seq*. *point-seq* is a sequence of points; *coord-seq* is a sequence of coordinate pairs, which are real numbers. It is an error if *coord-seq* does not contain an even number of elements.

If *closed?* is `#t`, then the segment connecting the first point and the last point is included in the polyline. The default for *closed?* is `#f`.

The function `make-polyline*` is identical to `make-polyline`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

	<i>Function</i>
<b>make-rectangle</b>	
Summary	Returns an object of class <code>&lt;rectangle&gt;</code> .
Signature	<pre><code>make-rectangle x1 y1 x2 y2 =&gt; rectangle</code></pre> <pre><code>make-rectangle* min-point max-point =&gt; rectangle</code></pre>
Arguments	The following arguments are specific to <code>make-rectangle</code> .
<i>x1</i>	An instance of type <code>&lt;real&gt;</code> . The <i>x</i> coordinate of the left top of the rectangle.

<i>y1</i>	An instance of type <code>&lt;real&gt;</code> . The <i>y</i> coordinate of the left top of the rectangle
<i>x2</i>	An instance of type <code>&lt;real&gt;</code> . The <i>x</i> coordinate of the bottom right of the rectangle.
<i>y2</i>	An instance of type <code>&lt;real&gt;</code> . The <i>y</i> coordinate of the bottom right of the rectangle.
The following arguments are specific to <code>make-rectangle*</code> .	
<i>min-point</i>	The minimum point (left top) of the rectangle.
<i>max-point</i>	The maximum point (bottom right) of the rectangle.
Values	<i>rectangle</i> An instance of type <code>&lt;rectangle&gt;</code> .
Description	<p>Returns an object of class <code>&lt;rectangle&gt;</code> whose edges are parallel to the coordinate axes. One corner is at the point <i>point1</i> or the position <i>x1,y1</i> and the opposite corner is at the point <i>point2</i> or the position <i>x2,y2</i>. There are no ordering constraints among <i>point1</i> and <i>point2</i> or <i>x1</i> and <i>x2</i>, and <i>y1</i> and <i>y2</i>.</p> <p>The function <code>make-rectangle*</code> is identical to <code>make-rectangle</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>

<b>&lt;polygon&gt;</b>		<i>Abstract instantiable class</i>
Summary	The class that corresponds to a polygon.	
Superclasses	<code>&lt;area&gt;</code>	
Init-keywords	<code>coordinates:</code>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;real&gt;)</code> .

	<b>points:</b> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;real&gt;)</code> .
Description	<p>The class that corresponds to a polygon. This is a subclass of <code>&lt;area&gt;</code>.</p> <p>A polygon can be described either in terms of the individual x and y coordinates that constitute its vertices, or by using composite points. If the former is used, then they can be specified at the time of creation using the <code>coordinates:</code> init-keyword, which is a sequence of real numbers, with x and y coordinates alternating within the sequence.</p> <p>To describe a polygon in terms of composite point objects, use the <code>points:</code> init-keyword, which is a sequence of instances of <code>&lt;point&gt;</code>. You should be aware that using composite points may lead to a loss of performance.</p> <p>Exactly one of <code>coordinates:</code> and <code>points:</code> is required.</p>
Operations	<p>The following operations are exported from the <b>DUIM-Extended-Geometry</b> module.</p> <pre>do-polygon-coordinates do-polygon-segments draw- design polygon? polygon-coordinates polygon-points</pre> <p>The following operations are exported from the <b>DUIM-Geometry</b> module.</p> <pre>box-edges transform-region</pre>
See also	<p><code>&lt;area&gt;</code>, page 10</p> <p><code>make-polygon</code>, page 83</p> <p><code>polygon?</code>, page 88</p> <p><code>polygon-coordinates</code>, page 88</p> <p><code>polygon-points</code>, page 89</p>

**polygon?** *Generic function*

Summary	Returns <code>#t</code> if its argument is a polygon.
Signature	<code>polygon? object =&gt; boolean</code>
Arguments	<i>object</i> An instance of type <code>&lt;object&gt;</code> .
Values	<i>boolean</i> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if <i>object</i> is a polygon, otherwise returns <code>#f</code> .
See also	<code>&lt;polygon&gt;</code> , page 86 <code>polygon-coordinates</code> , page 88 <code>polygon-points</code> , page 89

**polygon-coordinates** *Generic function*

Summary	Returns a sequence of coordinate pairs that specify the segments in a polygon or a polyline.
Signature	<code>polygon-coordinates polygon-or-polyline =&gt; coordinates</code>
Arguments	<i>polygon-or-polyline</i> An instance of type <code>type-union(&lt;polygon&gt;, &lt;polyline&gt;)</code> .
Values	<i>coordinates</i> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;real&gt;)</code> .
Description	Returns a sequence of coordinate pairs that specify the segments in <i>polygon-or-polyline</i> .
See also	<code>&lt;polygon&gt;</code> , page 86

`polygon?`, page 88

`polygon-points`, page 89

## **polygon-points**

### *Generic function*

Summary	Returns a sequence of points that specify the segments in a polygon or a polyline.
Signature	<code>polygon-points polygon-or-polyline =&gt; points</code>
Arguments	<code>polygon-or-polyline</code> An instance of type <code>type-union(&lt;polygon&gt;, &lt;polyline&gt;)</code> .
Values	<code>points</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;point&gt;)</code> .
Description	Returns a sequence of points that specify the segments in <code>polygon-or-polyline</code> .
See also	<code>&lt;polygon&gt;</code> , page 86 <code>polygon?</code> , page 88 <code>polygon-coordinates</code> , page 88

## **<polyline>**

### *Abstract instantiable class*

Summary	The protocol class that corresponds to a polyline.
Superclasses	<code>&lt;path&gt;</code>
Init-keywords	<code>coordinates:</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;real&gt;)</code> . Required.

**points:** An instance of type `limited(<sequence>, of: <real>)`. Required.

Description	<p>The protocol class that corresponds to a polyline.</p> <p>A polyline can be described either in terms of the individual x and y coordinates that constitute its vertices, or by using composite points. If the former is used, then they can be specified at the time of creation using the <code>coordinates:</code> init-keyword, which is a sequence of real numbers, with x and y coordinates alternating within the sequence.</p> <p>To describe a polyline in terms of composite point objects, use the <code>points:</code> init-keyword, which is a sequence of instances of <code>&lt;point&gt;</code>. You should be aware that using composite points may lead to a loss of performance.</p> <p>Exactly one of <code>coordinates:</code> and <code>points:</code> is required.</p>
Operations	<p>The following operations are exported from the <code>DUIM-Extended-Geometry</code> module.</p> <p><code>do-polygon-coordinates</code> <code>do-polygon-segments</code> <code>draw-design</code> <code>polygon-coordinates</code> <code>polygon-points</code> <code>polyline?</code> <code>polyline-closed?</code></p> <p>The following operations are exported from the <code>DUIM-Geometry</code> module.</p> <p><code>box-edges</code> <code>transform-region</code></p>
See also	<p><code>&lt;path&gt;</code>, page 40</p> <p><code>make-polyline</code>, page 84</p> <p><code>polyline?</code>, page 91</p> <p><code>polyline-closed?</code>, page 91</p>

**polyline?***Generic function*

Summary	Returns #t if an object is a polyline.	
Signature	<b>polyline? object =&gt; boolean</b>	
Arguments	<i>object</i>	An instance of type <object>.
Values	<i>boolean</i>	An instance of type <boolean>.
Description	Returns #t if <i>object</i> is a polyline, otherwise returns #f.	
See also	<a href="#">&lt;polyline&gt;, page 89</a> <a href="#">polyline-closed?, page 91</a>	

**polyline-closed?***Generic function*

Summary	Returns #t if the polyline is closed.	
Signature	<b>polyline-closed? polyline =&gt; boolean</b>	
Arguments	<i>polyline</i>	An instance of type <polyline>.
Values	<i>boolean</i>	An instance of type <boolean>.
Description	Returns #t if the polyline <i>polyline</i> is closed, otherwise returns #f. This function need be implemented only for polylines, not for polygons.	
See also	<a href="#">&lt;polyline&gt;, page 89</a> <a href="#">polyline?, page 91</a>	

<rectangle>		<i>Abstract instantiable class</i>
Summary	The protocol class that corresponds to a rectangle.	
Superclasses	<area>	
Init-keywords	<p><b>min-x:</b> An instance of type &lt;real&gt;.</p> <p><b>min-y:</b> An instance of type &lt;real&gt;.</p> <p><b>max-x:</b> An instance of type &lt;real&gt;.</p> <p><b>max-y:</b> An instance of type &lt;real&gt;.</p> <p><b>points:</b> An instance of type limited(&lt;sequence&gt;, of: &lt;point&gt;).</p>	
Description	<p>The protocol class that corresponds to a rectangle. This is a subclass of &lt;polygon&gt;.</p> <p>Rectangles whose edges are parallel to the coordinate axes are a special case of polygon that can be specified completely by four real numbers <math>x1,y1,x2,y2</math>). They are <i>not</i> closed under general affine transformations (although they are closed under rectilinear transformations).</p>	
Operations	<p>The following operations are exported from the <b>DUIM-Extended-Geometry</b> module.</p> <pre>do-polygon-coordinates do-polygon-segments draw- design polygon-coordinates polygon-points rectangle? rectangle-edges rectangle-height rectangle-max-point rectangle-max-position rectangle-min-point rectangle- min-position rectangle-size rectangle-width</pre> <p>The following operations are exported from the <b>DUIM-Geometry</b> module.</p> <pre>box-edges transform-region</pre>	
See also	<polygon>, page 86	

`make-rectangle`, page 85  
`rectangle?`, page 93  
`rectangle-edges`, page 94  
`rectangle-height`, page 95  
`rectangle-max-point`, page 96  
`rectangle-max-position`, page 96  
`rectangle-min-point`, page 97  
`rectangle-min-position`, page 98  
`rectangle-size`, page 99  
`rectangle-width`, page 99

<code>rectangle?</code>	<i>Generic function</i>
Summary	Returns #t if the object is a rectangle.
Signature	<code>rectangle? object =&gt; boolean</code>
Arguments	<code>object</code> An instance of type <object>.
Values	<code>boolean</code> An instance of type <boolean>.
Description	Returns #t if <i>object</i> is a rectangle, otherwise returns #f.
See also	<code>&lt;rectangle&gt;</code> , page 92 <code>rectangle-edges</code> , page 94 <code>rectangle-height</code> , page 95 <code>rectangle-max-point</code> , page 96 <code>rectangle-max-position</code> , page 96 <code>rectangle-min-point</code> , page 97

`rectangle-min-position`, page 98  
`rectangle-size`, page 99  
`rectangle-width`, page 99

## `rectangle-edges` *Generic function*

Summary	Returns the coordinates of the minimum and maximum of the rectangle.	
Signature	<code>rectangle-edges rectangle =&gt; x1 y1 x2 y2</code>	
Arguments	<code>rectangle</code>	An instance of type <code>&lt;rectangle&gt;</code> .
Values	<code>min-x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>min-y</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>max-x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>max-y</code>	An instance of type <code>&lt;real&gt;</code> .
Description	Returns the coordinates of the minimum <i>x</i> and <i>y</i> and maximum <i>x</i> and <i>y</i> of the rectangle <i>rectangle</i> as four values, <i>min-x</i> , <i>min-y</i> , <i>max-x</i> , and <i>max-y</i> .  The argument <i>min-x</i> represents the <i>x</i> coordinate of the top left of the rectangle.  The argument <i>min-y</i> represents the <i>y</i> coordinate of the top left of the rectangle.  The argument <i>max-x</i> represents the <i>x</i> coordinate of the bottom right of the rectangle.  The argument <i>max-y</i> represents the <i>y</i> coordinate of the bottom right of the rectangle.	
See also	<code>&lt;rectangle&gt;</code> , page 92 <code>rectangle?</code> , page 93	

**rectangle-height**, page 95  
**rectangle-max-point**, page 96  
**rectangle-max-position**, page 96  
**rectangle-min-point**, page 97  
**rectangle-min-position**, page 98  
**rectangle-size**, page 99  
**rectangle-width**, page 99

**rectangle-height** *Generic function*

Summary	Returns height of the rectangle.
Signature	<b>rectangle-height</b> <i>rectangle</i> => <i>height</i>
Arguments	<i>rectangle</i> An instance of type < <b>rectangle</b> >.
Values	<i>height</i> An instance of type < <b>real</b> >.
Description	Returns the height of the rectangle, which is the difference between the maximum y and its minimum y.
See also	<a href="#">&lt;rectangle&gt;</a> , page 92 <a href="#">rectangle?</a> , page 93 <a href="#">rectangle-edges</a> , page 94 <a href="#">rectangle-max-point</a> , page 96 <a href="#">rectangle-max-position</a> , page 96 <a href="#">rectangle-min-point</a> , page 97 <a href="#">rectangle-min-position</a> , page 98 <a href="#">rectangle-size</a> , page 99 <a href="#">rectangle-width</a> , page 99

## rectangle-max-point *Generic function*

Summary	Returns the bottom right point of the rectangle.
Signature	<code>rectangle-max-point <i>rectangle</i> =&gt; <i>point</i></code>
Arguments	<code><i>rectangle</i></code> An instance of type <code>&lt;rectangle&gt;</code> .
Values	<code><i>point</i></code> An instance of type <code>&lt;point&gt;</code> .
Description	Returns the bottom right point of the rectangle.
See also	<a href="#">`rectangle`</a> , page 92 <a href="#">`rectangle?`</a> , page 93 <a href="#">`rectangle-edges`</a> , page 94 <a href="#">`rectangle-height`</a> , page 95 <a href="#">`rectangle-max-position`</a> , page 96 <a href="#">`rectangle-min-point`</a> , page 97 <a href="#">`rectangle-min-position`</a> , page 98 <a href="#">`rectangle-size`</a> , page 99 <a href="#">`rectangle-width`</a> , page 99

## rectangle-max-position *Generic function*

Summary	Returns the <i>x</i> and <i>y</i> coordinates of the bottom right of the rectangle.
Signature	<code>rectangle-max-position <i>rectangle</i> =&gt; <i>x2 y2</i></code>
Arguments	<code><i>rectangle</i></code> An instance of type <code>&lt;rectangle&gt;</code> .
Values	<code><i>x2</i></code> An instance of type <code>&lt;real&gt;</code> .

<code>y2</code>	An instance of type <code>&lt;real&gt;</code> .
Description	Returns the <code>x</code> and <code>y</code> coordinates of the bottom right of the rectangle.
See also	<a href="#"><code>&lt;rectangle&gt;</code>, page 92</a> <a href="#"><code>rectangle?</code>, page 93</a> <a href="#"><code>rectangle-edges</code>, page 94</a> <a href="#"><code>rectangle-height</code>, page 95</a> <a href="#"><code>rectangle-max-point</code>, page 96</a> <a href="#"><code>rectangle-min-point</code>, page 97</a> <a href="#"><code>rectangle-min-position</code>, page 98</a> <a href="#"><code>rectangle-size</code>, page 99</a> <a href="#"><code>rectangle-width</code>, page 99</a>

## `rectangle-min-point` *Generic function*

Summary	Returns the left top point of the rectangle.	
Signature	<code>rectangle-min-point rectangle =&gt; point</code>	
Arguments	<code>rectangle</code>	An instance of type <code>&lt;rectangle&gt;</code> .
Values	<code>point</code>	An instance of type <code>&lt;point&gt;</code> .
Description	Returns the left top point of the rectangle.	
See also	<a href="#"><code>&lt;rectangle&gt;</code>, page 92</a> <a href="#"><code>rectangle?</code>, page 93</a> <a href="#"><code>rectangle-edges</code>, page 94</a> <a href="#"><code>rectangle-height</code>, page 95</a>	

`rectangle-max-point`, page 96  
`rectangle-max-position`, page 96  
`rectangle-min-position`, page 98  
`rectangle-size`, page 99  
`rectangle-width`, page 99

## **rectangle-min-position** *Generic function*

**Summary** Returns the *x* and *y* coordinates of the left top of the rectangle.

**Signature** `rectangle-min-position rectangle => x1 y1`

**Arguments** `rectangle` An instance of type `<rectangle>`.

**Values** `x1` An instance of type `<real>`.

`y1` An instance of type `<real>`.

**Description** Returns the *x* and *y* coordinates of the left top of the rectangle.

**See also** `<rectangle>`, page 92

`rectangle?`, page 93

`rectangle-edges`, page 94

`rectangle-height`, page 95

`rectangle-max-point`, page 96

`rectangle-max-position`, page 96

`rectangle-min-point`, page 97

`rectangle-size`, page 99

`rectangle-width`, page 99

**rectangle-size** *Generic function*

**Summary** Returns the width and the height of the rectangle.

**Signature** `rectangle-size rectangle => width height`

**Arguments** `rectangle` An instance of type `<rectangle>`.

**Values** `width` An instance of type `<real>`.

`height` An instance of type `<real>`.

**Description** Returns two values, the width and the height.

**See also** `<rectangle>`, page 92

`rectangle?`, page 93

`rectangle-edges`, page 94

`rectangle-height`, page 95

`rectangle-max-point`, page 96

`rectangle-max-position`, page 96

`rectangle-min-point`, page 97

`rectangle-min-position`, page 98

`rectangle-width`, page 99

**rectangle-width** *Generic function*

**Summary** Returns the width of the rectangle.

**Signature** `rectangle-width rectangle => width`

**Arguments** `rectangle` An instance of type `<rectangle>`.

**Values** `width` An instance of type `<real>`.

Description      Returns the width of the rectangle *rectangle*, which is the difference between the maximum *x* and its minimum *x*.

See also      `<rectangle>`, page 92

`rectangle?`, page 93

`rectangle-edges`, page 94

`rectangle-height`, page 95

`rectangle-max-point`, page 96

`rectangle-max-position`, page 96

`rectangle-min-point`, page 97

`rectangle-min-position`, page 98

`rectangle-size`, page 99

# 4

---

---

## DUIM-DCs Library

### 4.1 Overview

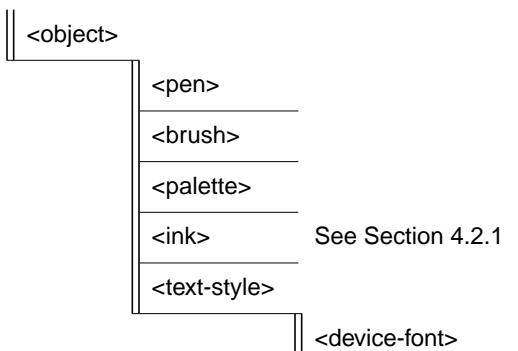
The DUIM-DCs library provides color support to the DUIM library. The library contains a single module, `duim-dcs`, from which all the interfaces described in this chapter are exposed. Section 4.3 on page 104 contains complete reference entries for each exposed interface.

Throughout this chapter, a *drawing context* consists of the combination of ink, color, brush, pen, palette, and shapes that make up patterns and images.

## 4.2 The class hierarchy for DUIM-DCs

A number of base classes are exposed in the DUIM-DCs library, each of which is a subclass of `<object>`. They are shown in Table 4.1.

**Table 4.1** Overall class hierarchy for the DUIM-DCs library

	
<code>&lt;pen&gt;</code>	This is protocol class for pens. A pen is used to draw 1 dimensional graphics such as lines or outline, using a specific color or pattern.
<code>&lt;brush&gt;</code>	The protocol class for brushes. Brushes are used to fill in 2 dimensional areas with a specific color or pattern.
<code>&lt;palette&gt;</code>	The protocol class for palettes. A palette provides a set of colors which can be made available to an application.
<code>&lt;ink&gt;</code>	This class can be thought of as anything that can be drawn. As the name implies, an ink describes the color and opacity features used by a given pen or brush. That is, the pen and brush define the drawing style (outlining or filling, respectively), and an ink is used to describe the color or pattern that is drawn. This class has a number of subclasses, described in Section 4.2.1.
<code>&lt;text-style&gt;</code>	The protocol class for text styles. A text style is a portable way of describing the appearance of a piece of text on screen (its font family, size, weight, and so on) in an abstract fashion. Because the fonts available on a particular computer may not necessarily match the fonts

available on the computer of the programmer, DUIM provides a portable model which allows the most suitable font on the user's machine to be chosen at run-time.

`<device-font>` The protocol class for device-specific fonts, that is, fonts that are resident on a particular device. This is a direct subclass of `<text-style>`.

### 4.2.1 Subclasses of `<ink>`

A number of subclasses of `<ink>` are exposed by the DUIM-DCs library, as follows:

- `<color>` The class of all colors available on the system. This is a direct subclass of `<ink>`.
- `<image>` The class of all images, such as icons and bitmap images. Images may often be acquired from an outside source, such as a file on disk. This is a direct subclass of `<ink>`.
- `<stencil>` A stencil is a special kind of pattern that contains only opacities, that is, it provides a layer of transparency. This can be useful, for instance, when overlaying a color onto an image, so as to provide the impression of shading. This is a direct subclass of `<image>`.
- `<pattern>` A pattern is a bounded rectangular arrangement of color, like a checkerboard. Drawing a pattern draws a different design in each rectangular cell of the pattern. This is a direct subclass of `<stencil>`.

### 4.2.2 Error classes provided by DUIM-DCs

Two error classes are provided by the DUIM-DCs library, both of which are immediate subclasses of `<error>`.

`<color-not-found>`

This class of error is signalled when a color is requested but is not available on the user's system.

**<palette-full>** This class of error is signalled when an attempt is made to add a color to a palette, and the palette cannot accept any more colors. The number of colors in a palette depends on the color depth of the connected monitor.

## 4.3 DUIM-DCs Module

This section contains a complete reference of all the interfaces that are exported from the `duim-dcs` module.

	<i>G.f. method</i>
=	
Summary	Returns <code>#t</code> if two objects are equal.
Signature	$= \text{color1} \text{ color2} \Rightarrow \text{boolean}$ $= \text{pen1} \text{ pen2} \Rightarrow \text{boolean}$ $= \text{brush1} \text{ brush2} \Rightarrow \text{boolean}$ $= \text{text-style1} \text{ text-style2} \Rightarrow \text{value}$
Arguments	$\text{color1}$ An instance of type <code>&lt;color&gt;</code> . $\text{color2}$ An instance of type <code>&lt;color&gt;</code> . $\text{pen1}$ An instance of type <code>&lt;pen&gt;</code> . $\text{pen2}$ An instance of type <code>&lt;pen&gt;</code> . $\text{brush1}$ An instance of type <code>&lt;brush&gt;</code> . $\text{brush2}$ An instance of type <code>&lt;brush&gt;</code> . $\text{text-style1}$ An instance of type <code>&lt;text-style&gt;</code> . $\text{text-style2}$ An instance of type <code>&lt;text-style&gt;</code> .
Values	$\text{boolean}$ An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if two objects are equal.

**add-colors** *Generic function*

Summary	Adds one or more colors to a palette and returns the updated palette.	
Signature	<code>add-colors palette #rest colors =&gt; palette</code>	
Arguments	<i>palette</i>	An instance of type <code>&lt;palette&gt;</code> .
	<i>colors</i>	Instances of type <code>&lt;color&gt;</code> .
Values	<i>palette</i>	An instance of type <code>&lt;palette&gt;</code> .
Description	Adds <i>colors</i> to <i>palette</i> and returns the updated palette.	

**\$background** *Constant*

Summary	An indirect ink that uses the medium's background design.	
Type	<code>&lt;ink&gt;</code>	
Description	An indirect ink that uses the medium's background design.	
See also	<a href="#">&lt;palette&gt;, page 150</a> <a href="#">image-height, page 137</a>	

**\$black** *Constant*

Summary	The usual definition of black.	
Type	<code>&lt;color&gt;</code>	
Description	The usual definition black, the absence of all colors. In the <code>rgb</code> color model, its value is <code>000</code> .	

See also [<color>](#), page 117

\$blue

## *Constant*

**Summary** The usual definition of the color blue.

Type <color>

Description The usual definition of the color blue.

See also [<color>](#), page 117

\$boole-clr

Constant

**Summary** The logical operator that is always 0.

Type <integer>

Description The logical operator that is always 0. It is a suitable first argument to the `bool` function.

\$boole-set

Constant

**Summary** The logical operator that is always 1.

Type <integer>

Description The logical operator that is always 1. It is a suitable first argument to the `bool` function.

## \$boole-1 *Constant*

Summary	The logical operator that is always the same as the first integer argument to the <code>boole</code> function.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is always the same as the first integer argument to the <code>boole</code> function. It is a suitable first argument to the <code>boole</code> function.

## \$boole-2 *Constant*

Summary	The logical operator that is always the same as the second integer argument to the <code>boole</code> function.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is always the same as the second integer argument to the <code>boole</code> function. It is a suitable first argument to the <code>boole</code> function.

## \$boole-c1 *Constant*

Summary	The logical operator that is always the same as the complement of the first integer argument to the <code>boole</code> function.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is always the same as the complement of the first integer argument to the <code>boole</code> function. It is a suitable first argument to the <code>boole</code> function.

<b>\$boole-c2</b>	<i>Constant</i>
Summary	The logical operator that is always the same as the complement of the second integer argument to the <code>boole</code> function.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is always the same as the complement of the second integer argument to the <code>boole</code> function. It is a suitable first argument to the <code>boole</code> function.
<b>\$boole-and</b>	<i>Constant</i>
Summary	The logical operator <code>and</code> .
Type	<code>&lt;integer&gt;</code>
Description	The logical operator <code>and</code> . It is a suitable first argument to the <code>boole</code> function.
<b>\$boole-ior</b>	<i>Constant</i>
Summary	The logical operator <code>inclusive or</code> .
Type	<code>&lt;integer&gt;</code>
Description	The logical operator <code>inclusive or</code> . It is a suitable first argument to the <code>boole</code> function.
<b>\$boole-xor</b>	<i>Constant</i>
Summary	The logical operator <code>exclusive or</code> .
Type	<code>&lt;integer&gt;</code>

Description      The logical operator **exclusive or**. It is a suitable first argument to the **boole** function.

### **\$boole-eqv** *Constant*

Summary      The logical operator **equivalence (exclusive nor)**.

Type            <integer>

Description      The logical operator **equivalence (exclusive nor)**. It is a suitable first argument to the **boole** function.

### **\$boole-nand** *Constant*

Summary      The logical operator **not-and**.

Type            <integer>

Description      The logical operator **not-and**. It is a suitable first argument to the **boole** function.

### **\$boole-nor** *Constant*

Summary      The logical operator **not-or**.

Type            <integer>

Description      The logical operator **not-or**. It is a suitable first argument to the **boole** function.

	<i>Constant</i>
<b>\$boole-andc1</b>	
Summary	The logical operator that is the <code>and</code> of the complement of the first integer argument to the <code>boole</code> function with the second.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is the <code>and</code> of the complement of the first integer argument to the <code>boole</code> function with the second. It is a suitable first argument to the <code>boole</code> function.
<b>\$boole-andc2</b>	
Summary	The logical operator that is the <code>and</code> of the first integer argument to the <code>boole</code> function with the second with the complement of the second.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is <code>and</code> of the first integer argument to the <code>boole</code> function with the complement of the second. It is a suitable first argument to the <code>boole</code> function.
<b>\$boole-orc1</b>	
Summary	The logical operator that is the <code>or</code> of the complement of the first integer argument to the <code>boole</code> function with the second.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is the <code>or</code> of the complement of the first integer argument to the <code>boole</code> function with the second. It is a suitable first argument to the <code>boole</code> function.

## \$boole-orc2 *Constant*

Summary	The logical operator that is the <code>or</code> of the first integer argument to the <code>boole</code> function with the second with the complement of the second.
Type	<code>&lt;integer&gt;</code>
Description	The logical operator that is <code>or</code> of the first integer argument to the <code>boole</code> function with the complement of the second. It is a suitable first argument to the <code>boole</code> function.

## \$bricks-stipple *Constant*

Summary	A stipple pattern for use in creating a patterned brush with horizontal and vertical lines in the pattern of the mortar in a brick wall.
Type	<code>&lt;array&gt;</code>
Description	A stipple pattern for use in creating a patterned brush with horizontal and vertical lines in the pattern of the mortar in a brick wall.
See also	<code>brush-stipple</code> , page 115

## <brush> *Abstract instantiable class*

Summary	The protocol class for brushes.
Superclasses	<code>&lt;object&gt;</code>
Init-keywords	<p><code>foreground</code>: An instance of type <code>&lt;ink&gt;</code>.</p> <p><code>background</code>: An instance of type <code>&lt;ink&gt;</code>.</p>

<b>mode:</b>	An instance of type <integer>.
<b>fill-style:</b>	An instance of type <b>false-or(&lt;integer&gt;)</b> . Default value: #f.
<b>fill-rule:</b>	An instance of type <b>false-or(&lt;integer&gt;)</b> . Default value: #f.
<b>tile:</b>	An instance of type <b>false-or(&lt;integer&gt;)</b> . Default value: #f.
<b>stipple:</b>	An instance of type <b>false-or(&lt;integer&gt;)</b> . Default value: #f.
<b>ts-x:</b>	An instance of <b>false-or(&lt;integer&gt;)</b> . Default value: #f.
<b>ts-y:</b>	An instance of <b>false-or(&lt;integer&gt;)</b> . Default value: #f.
Description	The protocol class for brushes.
Operations	<p>The following operations are exported from the <b>DUIM-DCs</b> module.</p> <pre>= brush? brush-background brush-fill-rule brush-fill-   style brush-foreground brush-mode brush-stipple brush-   stretch-mode brush-tile brush-ts-x brush-ts-y</pre>
See also	<b>make</b> , page 140

<b>brush?</b>		<i>Generic function</i>
Summary		Returns #t if its argument is a brush.
Signature		<b>brush? object =&gt; boolean</b>
Arguments	<i>object</i>	An instance of type <object>.
Values	<i>boolean</i>	An instance of type <boolean>.

Description     Returns `#t` if its argument is a brush.

## **brush-background** *Generic function*

Summary     Returns the ink that is the background color of a brush.

Signature     `brush-background brush => ink`

Arguments     *brush*        An instance of type `<brush>`.

Values        *ink*        An instance of type `<ink>`.

Description     Returns the *ink* that is the background color of *brush*.

See also     `brush-fill-rule`, page 113

## **brush-fill-rule** *Generic function*

Summary     Returns the fill rule of the brush.

Signature     `brush-fill-rule brush => fill-rule`

Arguments     *brush*        An instance of type `<brush>`.

Values        *fill-rule*        An instance of type `fill-rule` or `<boolean>`.

Description     Returns the fill rule for *brush*, or `#f` if *brush* does not have a fill rule.

See also     `brush-fill-style`, page 114

## brush-fill-style *Generic function*

Summary	Returns the fill style of the brush.	
Signature	<code>brush-fill-style brush =&gt; fill-style</code>	
Arguments	<i>brush</i>	An instance of type <code>&lt;brush&gt;</code> .
Values	<i>fill-style</i>	An instance of <code>fill-style</code> or <code>&lt;boolean&gt;</code> .
Description	Returns the fill style of <i>brush</i> , or <code>#f</code> , if <i>brush</i> does not have a fill style.	
See also	<a href="#">brush-fill-rule</a> , page 113.	

## brush-foreground *Generic function*

Summary	Returns the ink that is the foreground color of a brush.	
Signature	<code>brush-foreground brush =&gt; ink</code>	
Arguments	<i>brush</i>	An instance of type <code>&lt;brush&gt;</code> .
Values	<i>ink</i>	An instance of type <code>&lt;ink&gt;</code> .
Description	Returns the <i>ink</i> that is the foreground color of <i>brush</i> .	
See also	<a href="#">brush-stipple</a> , page 115.	

## brush-mode *Generic function*

Summary	Returns an integer representing the drawing mode of a brush.	
Signature	<code>brush-mode brush =&gt; integer</code>	

Arguments	<i>brush</i>	An instance of type <brush>.
Values	<i>integer</i>	An instance of type <integer>. Default value: \$boole-1.
Description		Returns an integer representing the drawing mode of <i>brush</i> .
See also		\$boole-1, page 107.

**brush-stipple***Generic function*

Summary		Returns the stipple pattern of a brush.
Signature		<b>brush-stipple</b> <i>brush</i> => <i>stipple</i>
Arguments	<i>brush</i>	An instance of type <brush>.
Values	<i>stipple</i>	A ( <i>stipple</i> ) or #f.
Description		Returns the stipple pattern of <i>brush</i> .
See also		<b>brush-tile</b> , page 116 <b>brush-fill-rule</b> , page 113 <b>brush-fill-style</b> , page 114

**brush-stretch-mode***Generic function*

Summary		Returns the stretch mode of the brush.
Signature		<b>brush-stretch-mode</b> <i>brush</i> => <i>stretch-mode</i>
Arguments	<i>brush</i>	An instance of type <brush>.
Values	<i>stretch-mode</i>	An instance of <i>stretch-mode</i> OR <boolean>.

Description      Returns the stretch mode of the brush.

### **brush-tile** *Generic function*

Summary      Returns the tile pattern of a brush.

Signature      **brush-tile** *brush* => *image*

Arguments      *brush*                  An instance of type <**brush**>.

Values            *image*                  An instance of type <**image**>.

Description      Returns the tile pattern of *brush*.

See also        **brush-stipple**, page 115.

**brush-ts-x**, page 116 and **brush-ts-y**, page 117.

### **brush-ts-x** *Generic function*

Summary      Returns the value of the *x* coordinate that is used to align the brush's tile or stipple pattern.

Signature      **brush-ts-x** *brush* => *value*

Arguments      *brush*                  An instance of type <**brush**>.

Values            *value*                  An instance of type **false-or(<integer>)**.

Description      Returns the value of the *x* coordinate that is used to align the tile or stipple pattern of *brush*. If *brush* has no tile or stipple pattern, **brush-ts-x** returns #f.

See also        **brush-ts-y**, page 117.

## **brush-ts-y** *Generic function*

Summary	Returns the value of the y coordinate that is used to align the brush's tile or stipple pattern.	
Signature	<b>brush-ts-y</b>	<i>brush =&gt; value</i>
Arguments	<i>brush</i>	An instance of type <code>&lt;brush&gt;</code> .
Values	<i>value</i>	An instance of type <code>false-or(&lt;integer&gt;)</code> .
Description	Returns the value of the y coordinate that is used to align the tile or stipple pattern of <i>brush</i> . If <i>brush</i> has no tile or stipple pattern, <b>brush-ts-y</b> returns <code>#f</code> .	
See also	<b>brush-ts-x</b> , page 116.	

## **<color>** *Abstract instantiable class*

Summary	The protocol class for colors.	
Superclasses	<code>&lt;ink&gt;</code>	
Init-keywords	<b>red:</b>	An instance of type <code>&lt;real&gt;</code> .
	<b>green:</b>	An instance of type <code>&lt;real&gt;</code> .
	<b>blue:</b>	An instance of type <code>&lt;real&gt;</code> .
	<b>intensity:</b>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: sqrt(3))</code> .
	<b>hue:</b>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
	<b>saturation:</b>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
	<b>opacity:</b>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .

Description	<p>The <code>&lt;color&gt;</code> class is the protocol class for a color, and is a subclass of <code>&lt;ink&gt;</code>. A member of the class <code>&lt;color&gt;</code> is an ink that represents the intuitive definition of color: white, black, red, pale yellow, and so forth. The visual appearance of a single point is completely described by its color. Drawing a color sets the color of every point in the drawing plane to that color, and sets the opacity to 1.</p> <p>The <code>red:</code>, <code>green:</code>, and <code>blue:</code> init-keywords represent the red, green, and blue components of the color. For an 8-bit color scheme, these can take any real number in the range 0 to 255.</p> <p>The intensity describes the brightness of the color. An intensity of 0 is black.</p> <p>The hue of a color is the characteristic that is represented by a name such as red, green, blue and so forth. This is the main attribute of a color that distinguishes it from other colors.</p> <p>The saturation describes the amount of white in the color. This is what distinguishes pink from red.</p> <p>Opacity controls how new color output covers previous color output (that is, the final appearance when one color is painted on top of another). Opacity can vary from totally opaque (a new color completely obliterates the old color) to totally transparent (a new color has no effect whatsoever; the old color remains unchanged). Intermediate opacity values result in color blending so that the earlier color shows through what is drawn on top of it.</p> <p>All of the standard instantiable color classes provided by DUIM are immutable.</p> <p>A color can be specified by four real numbers between 0 and 1 (inclusive), giving the amounts of red, green, blue, and opacity (<i>alpha</i>). Three 0's for the RGB components mean black; three 1's mean white. The intensity-hue-saturation color model is also supported, but the red-green-blue color model is the primary model we will use in the specification.</p>
-------------	--

An opacity may be specified by a real number between 0 and 1 (inclusive). 0 is completely transparent, 1 is completely opaque, fractions are translucent. The opacity of a color is the degree to which it hides the previous contents of the drawing plane when it is drawn.

## Operations

The following operations are exported from the **DUIM-DCs** module.

**= color?** **color-rgb** **color-ihs** **color-luminosity**

## See also

**color?**, page 119  
**color-ihs**, page 120  
**color-luminosity**, page 121  
**<color-not-found>**, page 121  
**color-palette?**, page 122  
**color-rgb**, page 122  
**<ink>**, page 138

## color?

*Generic function*

### Summary

Returns #t if object is a color.

### Signature

**color? object => boolean**

### Arguments

**object** An instance of type **<object>**.

### Values

**boolean** An instance of type **<boolean>**.

### Description

Returns #t if object is a color, otherwise returns #f.

### See also

**<color>**, page 117  
**color-ihs**, page 120

`color-luminosity`, page 121  
`<color-not-found>`, page 121  
`color-palette?`, page 122  
`color-rgb`, page 122

	<i>Generic function</i>
<b>color-ihs</b>	
Summary	Returns four values, the intensity, hue, saturation, and opacity components of a color.
Signature	<code>color-ihs color =&gt; intensity hue saturation opacity</code>
Arguments	<code>color</code> An instance of type <code>&lt;color&gt;</code> .
Values	<code>intensity</code> An instance of type <code>limited(&lt;real&gt;, min: 0, max: sqrt(3))</code> . <code>hue</code> An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> . <code>saturation</code> An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> . <code>opacity</code> An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
Description	Returns four values, the <i>intensity</i> , <i>hue</i> , <i>saturation</i> , and <i>opacity</i> components of the color <code>color</code> . The first value is a real number between 0 and <code>sqrt{3}</code> (inclusive). The second and third values are real numbers between 0 and 1 (inclusive).
See also	<code>&lt;color&gt;</code> , page 117 <code>color?</code> , page 119 <code>color-luminosity</code> , page 121 <code>color-palette?</code> , page 122

`color-rgb`, page 122

## color-luminosity *Generic function*

Summary	Returns the brightness of a color.	
Signature	<code>color-luminosity color =&gt; luminosity</code>	
Arguments	<code>color</code>	An instance of type <code>&lt;color&gt;</code> .
Values	<code>luminosity</code>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
Description	Returns the brightness of color <code>color</code> as real number between 0 and 1. The value is the solution of a function that describes the perception of the color by the human retina.	
See also	<a href="#">&lt;color&gt;</a> , page 117 <a href="#">color?</a> , page 119 <a href="#">color-ihs</a> , page 120 <a href="#">color-palette?</a> , page 122 <a href="#">color-rgb</a> , page 122	

## <color-not-found> *Sealed concrete class*

Summary	The class of the error that is signalled when a color that is not available is requested.	
Superclasses	<code>&lt;error&gt;</code>	
Superclasses	<code>&lt;error&gt;</code>	
Init-keywords	<code>color:</code>	An instance of type <code>&lt;color&gt;</code> .

Description	The class of the error that is signalled when a color that is not available is requested. The <code>color:</code> init-keyword is used to specify the color that was requested but was not available.
Operations	None.
See also	<code>&lt;color&gt;</code> , page 117 <code>find-color</code> , page 131 <code>remove-colors</code> , page 160

## **color-palette?**

*Generic function*

Summary      Returns `#t` if the stream or medium supports color.

Signature      `color-palette? palette => boolean`

Arguments      `palette`      An instance of type `<palette>`.

Values      `boolean`      An instance of type `<boolean>`.

Description      Returns `#t` if the stream or medium supports color.

See also      `<color>`, page 117

`color?`, page 119

`color-ihs`, page 120

`color-luminosity`, page 121

`color-rgb`, page 122

## **color-rgb**

*Generic function*

Summary      Returns four values, the red, green, blue, and opacity components of a color.

Signature	<code>color-rgb color =&gt; ref green blue opacity</code>	
Arguments	<code>color</code>	An instance of type <code>&lt;color&gt;</code> .
Values	<code>red</code>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code>
	<code>green</code>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code>
	<code>blue</code>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code>
	<code>opacity</code>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
Description	Returns four values, the <code>red</code> , <code>green</code> , <code>blue</code> , and <code>opacity</code> components of the color <code>color</code> . The values are real numbers between 0 and 1 (inclusive).	
See also	<a href="#">`&lt;color&gt;`</a> , page 117 <a href="#">`color?`</a> , page 119 <a href="#">`color-ihs`</a> , page 120 <a href="#">`color-luminosity`</a> , page 121 <a href="#">`color-palette?`</a> , page 122	

## contrasting-colors-limit

*Generic function*

Summary	Returns the number of contrasting colors that can be rendered on the current platform.	
Signature	<code>contrasting-colors-limit port =&gt; integer</code>	
Arguments	<code>port</code>	An instance of type <code>&lt;port&gt;</code> .
Values	<code>integer</code>	An instance of type <code>&lt;integer&gt;</code> .

Description	Returns the number of contrasting colors (or stipple patterns if port is monochrome or grayscale) that can be rendered on any medium on the port <i>port</i> . Implementations are encouraged to make this as large as possible, but it must be at least 8. All classes that obey the medium protocol must implement a method for this generic function.
See also	<code>contrasting-dash-patterns-limit</code> , page 124 <code>make-contrasting-colors</code> , page 142

## **contrasting-dash-patterns-limit** *Generic function*

Summary	Returns the number of contrasting dash patterns that the specified port can generate.
Signature	<code>contrasting-dash-patterns-limit port =&gt; no-of-patterns</code>
Arguments	<i>port</i> An instance of type <code>&lt;port&gt;</code> .
Values	<i>no-of-patterns</i> An instance of type <code>&lt;integer&gt;</code> .
Description	Returns the number of contrasting dash patterns that the specified port can generate.
See also	<code>contrasting-colors-limit</code> , page 123 <code>make-contrasting-dash-patterns</code> , page 143

## **\$cross-hatch** *Constant*

Summary	A stipple pattern for use in creating a patterned brush with alternating solid and dashed lines.
Type	<code>&lt;array&gt;</code>

Description A stipple pattern for use in creating a patterned brush with alternating solid and dashed lines.

See also [<color>](#), page 117.

## \$cyan *Constant*

Summary The usual definition for the color cyan.

Type [<color>](#)

Description The usual definition for the color cyan.

See also [<color>](#), page 117.

## \$dash-dot-dot-pen *Constant*

Summary A pen that draws a line with two dots between each dash.

Type [<pen>](#)

Description A pen that draws a line with two dots between each dash.  
The line width is 1 and `dashes:` is #[4, 1, 1, 1, 1].

See also [<pen>](#), page 153

[\\$solid-pen](#), page 160

[\\$magenta](#), page 139

[\\$dash-dot-pen](#), page 126

[\\$dotted-pen](#), page 131

**\$dash-dot-pen** *Constant*

Summary A pen that draws a dashed and dotted line.

Type `<pen>`

Description A pen that draws a dashed and dotted line. The line width is `1` and `dashes:` is `#*[4, 1, 1, 1]`.

See also `<pen>`, page 153

`$solid-pen`, page 160

`$magenta`, page 139

`$dash-dot-pen`, page 126

`$dotted-pen`, page 131

**\$dashed-pen** *Constant*

Summary A pen that draws a dashed line.

Type `<pen>`

Description A pen that draws a dashed line. The line width is `1` and `dashes:` is `#t`.

See also `<pen>`, page 153

`$solid-pen`, page 160

`$magenta`, page 139

`$dash-dot-pen`, page 126

`$dotted-pen`, page 131

## **default-background** *Generic function*

Summary	Returns the ink that is the default background of its argument.	
Signature	<b>default-foreground</b> <i>object</i> => <i>background</i>	
Arguments	<i>object</i>	An instance of type < <i>object</i> >.
Values	<i>background</i>	An instance of type < <i>ink</i> >.
Description	Returns the ink that is the default background of its argument.	
See also	<a href="#">brush-fill-style</a> , page 114. <a href="#">default-background-setter</a> , page 127. <a href="#">default-foreground</a> , page 128	

## **default-background-setter** *Generic function*

Summary	Sets the default background.	
Signature	<b>default-foreground-setter</b> <i>background object</i> => <i>background</i>	
Arguments	<i>background</i>	An instance of type < <i>ink</i> >.
	<i>object</i>	An instance of type < <i>object</i> >.
Values	<i>background</i>	An instance of type < <i>ink</i> >.
Description	Sets the default background for <i>object</i> .	
See also	<a href="#">brush-fill-style</a> , page 114. <a href="#">default-background</a> , page 127. <a href="#">default-foreground-setter</a> , page 128	

**default-foreground** *Generic function*

Summary	Returns the ink that is the default foreground of its argument.	
Signature	<code>default-foreground object =&gt; foreground</code>	
Arguments	<i>object</i>	An instance of type <object>.
Values	<i>foreground</i>	An instance of type <ink>.
Description	Returns the ink that is the default foreground of its argument.	
See also	<code>brush-fill-rule</code> , page 113. <code>default-background</code> , page 127 <code>default-foreground-setter</code> , page 128	

**default-foreground-setter** *Generic function*

Summary	Sets the default foreground.	
Signature	<code>default-foreground-setter foreground object =&gt; foreground</code>	
Arguments	<i>foreground</i>	An instance of type <ink>.
	<i>object</i>	An instance of type <object>.
Values	<i>foreground</i>	An instance of type <ink>.
Description	Sets the default foreground for <i>object</i> .	
See also	<code>brush-fill-rule</code> , page 113. <code>default-background-setter</code> , page 127 <code>default-foreground</code> , page 128	

**default-text-style** *Generic function*

Summary	Returns the default text style for its argument.	
Signature	<code>default-text-style object =&gt; text-style</code>	
Arguments	<i>object</i>	An instance of type < <code>object</code> >.
Values	<i>text-style</i>	An instance of type < <code>text-style</code> >.
Description	Returns the default text style for its argument. This function is used to merge against if the text style is not fully specified, or if no text style is specified.	
See also	<a href="#">default-text-style-setter</a> , page 129.	

**default-text-style-setter** *Generic function*

Summary	Sets the default text style.	
Signature	<code>default-text-style-setter text-style object =&gt; text-style</code>	
Arguments	<i>text-style</i>	An instance of type < <code>text-style</code> >.
	<i>object</i>	An instance of type < <code>object</code> >.
Values	<i>text-style</i>	An instance of type < <code>text-style</code> >.
Description	Sets the default text style.	
See also	<a href="#">default-text-style</a> , page 129	

**<device-font>** *Sealed concrete class*

Summary	The protocol class for device-specific fonts.
---------	---

Superclasses    `<text-style>`

Init-keywords    `port:`

`font-name:`

Description    The protocol class for device-specific fonts.

Operations    None.

See also        `<text-style>`, page 162.

## \$diagonal-hatch-down

*Constant*

Summary    A stipple pattern for use in creating a patterned brush with alternating dashes and spaces.

Type        `<array>`

Description    A stipple pattern for use in creating a patterned brush with alternating dashes and spaces, the first line starting with a dash, followed by a space, and the second line starting with a space followed by a dash.

See also      `brush-stipple`, page 115.

## \$diagonal-hatch-up

*Constant*

Summary    A stipple pattern for use in creating a patterned brush with alternating dashes and spaces.

Type        `<array>`

Description A stipple pattern for use in creating a patterned brush with alternating dashes and spaces, the first line starting with a space, followed by a dash, and the second line starting with a dash followed by a space.

See also [brush-stipple](#), page 115.

## \$dotted-pen *Constant*

Summary A pen that draws a dotted line.

Type `<pen>`

Description A pen that draws a dotted line. The line width is 1 and `dashes:` is `#![1, 1]`.

See also [<pen>](#), page 153

[\\$solid-pen](#), page 160

[\\$dash-dot-pen](#), page 126

## find-color *Generic function*

Summary Looks up and returns a color by name.

Signature `find-color name palette #key error? => color`

Arguments `name` An instance of type `<string>`.

`palette` An instance of type `<palette>`.

`error?` An instance of type `<boolean>`. Default value: `#f`.

Values `color` An instance of type `<color>`.

Description      Looks up and returns a color by name. Table 4.2 lists the commonly provided color names that can be looked up with `find-color`.

alice-blue	antique-white	aquamarine
azure	beige	bisque
black	blanched-almond	blue
blue-violet	brown	burlywood
cadet-blue	chartreuse	chocolate
coral	cornflower-blue	cornsilk
cyan	dark-goldenrod	dark-green
dark-khaki	dark-olive-green	dark-orange
dark-orchid	dark-salmon	dark-sea-green
dark-slate-blue	dark-slate-gray	dark-turquoise
dark-violet	deep-pink	deep-sky-blue
dim-gray	dodger-blue	firebrick
floral-white	forest-green	gainsboro
ghost-white	gold	goldenrod
gray	green	green-yellow
honeydew	hot-pink	indian-red
ivory	khaki	lavender
lavender-blush	lawn-green	lemon-chiffon
light-blue	light-coral	light-cyan
light-goldenrod	light-goldenrod-yellow	light-gray
light-pink	light-salmon	light-sea-green
light-sky-blue	light-slate-blue	light-slate-gray
light-steel-blue	light-yellow	lime-green

Table 4.2 Common color names

linen	magenta	maroon
medium-aquamarine	medium-blue	medium-orchid
medium-purple	medium-sea-green	medium-slate-blue
medium-spring-green	medium-turquoise	medium-violet-red
midnight-blue	mint-cream	misty-rose
moccasin	navajo-white	navy-blue
old-lace	olive-drab	orange
orange-red	orchid	pale-goldenrod
pale-green	pale-turquoise	pale-violet-red
papaya-whip	peach-puff	peru
pink	plum	powder-blue
purple	red	rosy-brown
royal-blue	saddle-brown	salmon
sandy-brown	sea-green	seashell
sienna	sky-blue	slate-blue
slate-gray	snow	spring-green
steel-blue	tan	thistle
tomato	turquoise	violet
violet-red	wheat	white
white-smoke	yellow	yellow-green

**Table 4.2** Common color names

Application programs can define other colors; these are provided because they are commonly used in the X Windows community, not because there is anything special about these particular colors.

See also      `$black`, page 105

`stencil?`, page 161

**\$red**, page 159  
**\$yellow**, page 171  
**\$green**, page 135  
**\$blue**, page 106  
**\$magenta**, page 139  
**contrasting-dash-patterns-limit**, page 124

**\$foreground** *Constant*

Summary An indirect ink that uses the medium's foreground design.  
Type **<ink>**  
Description An indirect ink that uses the medium's foreground design.  
See also **<ink>**, page 138  
**<palette>**, page 150

**fully-merged-text-style?** *Generic function*

Summary Returns #t if the specified text style is completely specified.  
Signature **fully-merged-text-style? text-style => boolean**  
Arguments **text-style** An instance of type **<text-style>**.  
Values **boolean** An instance of type **<boolean>**.  
Description Returns #t if the specified text style is completely specified.  
See also **merge-text-styles**, page 150

**\$green** *Constant*

Summary The usual definition of the color green.

Type `<color>`

Description The usual definition of the color green.

See also `<color>`, page 117

**\$hearts-stipple** *Constant*

Summary A stipple pattern for use in creating a patterned brush that draws a heart shape.

Type `<array>`

Description A stipple pattern for use in creating a patterned brush that draws a heart shape.

See also `brush-stipple`, page 115

**\$horizontal-hatch** *Constant*

Summary A stipple pattern for use in creating a patterned brush with alternating horizontal rows of lines and spaces.

Type `<array>`

Description A stipple pattern for use in creating a patterned brush with alternating horizontal rows of lines and spaces.

See also `brush-stipple`, page 115.

<b>&lt;image&gt;</b>		<i>Abstract class</i>
Summary	The class for objects that are images.	
Superclasses	<code>&lt;ink&gt;</code>	
Init-keywords	None.	
Description	The class for objects that are images.	
Operations	The following operation is exported from the <code>DUIM-DCs</code> module.  <code>image?</code>  The following operation is exported from the <code>DUIM-Graphics</code> module.  <code>draw-image</code>	
See also	<code>image?</code> , page 136  <code>image-depth</code> , page 137  <code>image-height</code> , page 137  <code>image-width</code> , page 138  <code>&lt;ink&gt;</code> , page 138	

<b>image?</b>		<i>Generic function</i>
Summary	Returns <code>#t</code> if its argument is an image.	
Signature	<code>image? object =&gt; boolean</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .

Description      Returns `#t` if its argument is an image.

See also      `<image>`, page 136

`image-depth`, page 137

`image-height`, page 137

`image-width`, page 138

## image-depth

*Generic function*

Summary      Returns the depth of an image.

Signature      `image-depth image => depth`

Arguments      `image`      An instance of type `<image>`.

Values      `depth`      An instance of type `<real>`.

Description      Returns the depth of the image `image`.

See also      `<image>`, page 136

`image?`, page 136

`image-height`, page 137

`image-width`, page 138

## image-height

*Generic function*

Summary      Returns the height of an image.

Signature      `image-height image => height`

Arguments      `image`      An instance of type `<image>`.

Values	<i>height</i>	An instance of type <real>.
Description	Returns the height of the image <i>image</i> .	
See also	<i>&lt;image&gt;</i> , page 136 <i>image?</i> , page 136 <i>image-depth</i> , page 137 <i>image-width</i> , page 138	

## **image-width** *Generic function*

Summary	Returns the width of an image.	
Signature	<b>image-width</b> <i>image</i> => <i>width</i>	
Arguments	<i>image</i>	An instance of type <image>.
Values	<i>width</i>	An instance of type <real>.
Description	Returns the width of the image <i>image</i> .	
See also	<i>&lt;image&gt;</i> , page 136 <i>image?</i> , page 136 <i>image-depth</i> , page 137 <i>image-height</i> , page 137	

## **<ink>** *Abstract class*

Summary	The class of objects that represent a way of arranging colors and opacities in the drawing plane.
Superclasses	<object>

Init-keywords	None.
Description	The class of objects that represent a way of arranging colors and opacities in the drawing plane. Intuitively, it is anything that can be drawn with. An ink is anything that can be used in medium-foreground, medium-background, medium-ink, or the foreground or background of a brush.
Operations	The following operation is exported from the <code>DUIM-DCS</code> module.  <code>ink?</code>
See also	<code>ink?</code> , page 139

**ink?***Generic function*

Summary	Returns <code>#t</code> if its argument is an ink.
Signature	<code>ink? object =&gt; boolean</code>
Arguments	<code>object</code> An instance of type <code>&lt;object&gt;</code> .
Values	<code>boolean</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if <code>object</code> is an ink, otherwise returns <code>#f</code> .
See also	<code>&lt;ink&gt;</code> , page 138

**\$magenta***Constant*

Summary	The usual definition of the color magenta.
Type	<code>&lt;color&gt;</code>

Description      The usual definition of the color magenta.

See also      `<color>`, page 117

## **make** *G.f. method*

Summary      Returns an object that is of the same type as the class given as its argument.

Signature      `make (class == <pen>) #key width units dashes joint-shape cap-shape => pen`  
`make (class == <brush>) #key foreground background mode fill-style fill-rule tile stipple ts-x ts-y => brush`

Arguments      (`class==<pen>`) The class `<pen>`.

`width`      An instance of type `<pen-width>`. Default value: 1.

`units`      An instance of type `<pen-units>`. Default value: `#"normal"`.

`dashes`      An instance of type `<pen-dashes>`. Default value: `#f`.

`joint-shape`      An instance of type `<pen-joint-shape>`. Default value: `#"miter"`.

`cap-shape`      An instance of type `<pen-cap-shape>`. Default value: `#"butt"`.

(`class==<brush>`) The class `<brush>`.

`foreground`      An instance of type `<ink>`. Default value: `$foreground`.

`background`      An instance of type `<ink>`. Default value: `$background`.

`mode`      An instance of type `<integer>`. Default value: `$boolean-1`.

	<i>fill-style</i>	A ( <code>fill-style</code> ) or #f. Default value: #f.
	<i>fill-rule</i>	A ( <code>fill-rule</code> ) or #f. Default value: #f.
	<i>tile</i>	An ( <code>image</code> ) or #f. Default value: #f.
	<i>stipple</i>	A ( <code>stipple</code> ) or #f. Default value: #f.
	<i>ts-x</i>	An instance of <code>false-or(&lt;integer&gt;)</code> . Default value: #f.
	<i>ts-y</i>	An instance of <code>false-or(&lt;integer&gt;)</code> . Default value: #f.
Values	<i>pen</i>	An instance of type <code>&lt;pen&gt;</code> .
	<i>brush</i>	An instance of type <code>&lt;brush&gt;</code> .
Description		Returns an object that is of the same type as the class given as its argument. Default values for the keywords that specify object are provided, or the keywords can be given explicitly to override the defaults.
See also		<code>&lt;brush&gt;</code> , page 111 <code>&lt;pen&gt;</code> , page 153

## make-color-for-contrasting-color *Generic function*

Summary	Returns a color that is recognizably different from the main color.	
Signature	<code>make-color-for-contrasting-color ink =&gt; color</code>	
Arguments	<i>ink</i>	An instance of type <code>&lt;ink&gt;</code> .
Values	<i>color</i>	An instance of type <code>&lt;color&gt;</code> .
Description	Returns a color that is recognizably different from the main color.	

See also [make-contrasting-colors](#), page 142

	<b>make-contrasting-colors</b>	<i>Function</i>
Summary	Returns a vector of colors with recognizably different appearance.	
Signature	<code>make-contrasting-colors n #key k =&gt; colors</code>	
Arguments	<p><i>n</i> An instance of type <code>&lt;integer&gt;</code>.</p> <p><i>k</i> An instance of type <code>&lt;integer&gt;</code>.</p>	
Values	<i>colors</i> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;color&gt;)</code> .	
Description	<p>Returns a vector of <i>n</i> colors with recognizably different appearance. Elements of the vector are guaranteed to be acceptable values for the <code>brush:</code> argument to the drawing functions, and do not include <code>\$foreground</code>, <code>\$background</code>, or <code>nil</code>. Their class is otherwise unspecified. The vector is a fresh object that may be modified.</p> <p>If <i>k</i> is supplied, it must be an integer between 0 and <i>n</i> - 1 (inclusive), in which case <code>make-contrasting-colors</code> returns the <i>k</i>th color in the vector rather than the whole vector.</p> <p>If the implementation does not have <i>n</i> different contrasting colors, <code>make-contrasting-colors</code> signals an error. This does not happen unless <i>n</i> is greater than eight.</p> <p>The rendering of the color is a true color or a stippled pattern, depending on whether the output medium supports color.</p>	
See also	<a href="#">contrasting-colors-limit</a> , page 123 <a href="#">\$green</a> , page 135 <a href="#">make-color-for-contrasting-color</a> , page 141	

**make-contrasting-dash-patterns**, page 143

	<i>Function</i>				
Summary	Returns a vector of dash patterns with recognizably different appearances.				
Signature	<b>make-contrasting-dash-patterns</b> <i>n</i> #key <i>k</i> => <i>dashes</i>				
Arguments	<table border="0"> <tr> <td><i>n</i></td><td>An instance of type &lt;integer&gt;.</td></tr> <tr> <td><i>k</i></td><td>An instance of type &lt;integer&gt;.</td></tr> </table>	<i>n</i>	An instance of type <integer>.	<i>k</i>	An instance of type <integer>.
<i>n</i>	An instance of type <integer>.				
<i>k</i>	An instance of type <integer>.				
Values	<i>dashes</i> An instance of type <vector>.				
Description	<p>Returns a vector of <i>n</i> dash patterns with recognizably different appearances. If the keyword <i>k</i> is supplied, <b>make-contrasting-dash-patterns</b> returns the <i>k</i>th pattern. If there are not <i>n</i> different dash patterns, an error is signalled.</p> <p>The argument <i>n</i> represents the number of dash patterns.</p> <p>The argument <i>k</i> represents the index in the vector of dash patterns indicating the pattern to use.</p>				
See also	<a href="#">contrasting-dash-patterns-limit</a> , page 124 <a href="#">make-contrasting-colors</a> , page 142				

	<i>Function</i>				
Summary	Returns a device-specific font.				
Signature	<b>make-device-font</b> <i>port</i> <i>font</i> => <i>device-font</i>				
Arguments	<table border="0"> <tr> <td><i>port</i></td><td>An instance of type &lt;port&gt;.</td></tr> <tr> <td><i>font</i></td><td>An instance of type &lt;object&gt;.</td></tr> </table>	<i>port</i>	An instance of type <port>.	<i>font</i>	An instance of type <object>.
<i>port</i>	An instance of type <port>.				
<i>font</i>	An instance of type <object>.				

Values	<i>device-font</i>	A font object or the name of a font.
Description	Returns a device-specific font. Text styles are mapped to fonts for a port, a character set, and a text style. All ports must implement methods for the generic functions, for all classes of text style.	The objects used to represent a font mapping are unspecified and are likely to vary from port to port. For instance, a mapping might be some sort of font object on one type of port, or might simply be the name of a font on another.
		Part of initializing a port is to define the mappings between text styles and font names for the port's host window system.

		<i>Function</i>
make-gray-color		
Summary		Returns a member of class <color>.
Signature		<code>make-gray-color <i>luminosity</i> #key <i>opacity</i> =&gt; <i>color</i></code>
Arguments	<i>luminosity</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
	<i>opacity</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> . Default value: 1.0.
Values	<i>color</i>	An instance of type <color>.
Description		Returns a member of class <color>. The <i>luminance</i> is a real number between 0 and 1 (inclusive). On a black-on-white display device, 0 means black, 1 means white, and the values in between are shades of gray. On a white-on-black display device, 0 means white, 1 means black, and the values in between are shades of gray.
See also		<code>make-ihs-color</code> , page 145

[make-rgb-color](#), page 147

## make-ihs-color *Function*

Summary	Returns a member of the class <color>.	
Signature	<code>make-ihs-color <i>intensity hue saturation</i> #key <i>opacity</i> =&gt; <i>color</i></code>	
Arguments	<i>intensity</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: sqrt(3))</code> .
	<i>hue</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
	<i>saturation</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .
	<i>opacity</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> . Default value: 1.0.
Values	<i>color</i>	An instance of type <color>.
Description	Returns a member of class <color>. The <i>intensity</i> argument is a real number between 0 and $\sqrt{3}$ (inclusive). The <i>hue</i> and <i>saturation</i> arguments are real numbers between 0 and 1 (inclusive).	
See also	<a href="#">make-gray-color</a> , page 144 <a href="#">make-rgb-color</a> , page 147	

## make-palette *Generic function*

Summary	Returns a member of the class <palette>.
Signature	<code>make-palette <i>port</i> #key =&gt; <i>palette</i></code>

Arguments	<i>port</i>	An instance of type < <b>port</b> >.
Values	<i>palette</i>	An instance of type < <b>palette</b> >.
Description		Returns a member of the class < <b>palette</b> >.

		<i>Function</i>
	<b>make-pattern</b>	
Summary		Returns a pattern generated from a two-dimensional array.
Signature		<b>make-pattern</b> <i>array colors =&gt; pattern</i>
Arguments	<i>array</i>	An instance of type < <b>array</b> >.
	<i>colors</i>	An instance of type <b>limited(&lt;sequence&gt;, of: &lt;color&gt;)</b> .
Values	<i>pattern</i>	An instance of type < <b>pattern</b> >.
Description		Returns a pattern design that has ( <b>array-dimension array 0</b> ) cells in the vertical direction and ( <b>array-dimension array 1</b> ) cells in the horizontal direction. <i>array</i> must be a two-dimensional array of non-negative integers less than the length of <i>designs</i> . <i>designs</i> must be a sequence of designs. The design in cell <i>i,j</i> of the resulting pattern is the <i>n</i> th element of <i>designs</i> , if <i>n</i> is the value of ( <b>aref array i j</b> ). For example, <i>array</i> can be a bit-array and <i>designs</i> can be a list of two designs, the design drawn for 0 and the one drawn for 1. Each cell of a pattern can be regarded as a hole that allows the design in it to show through. Each cell might have a different design in it. The portion of the design that shows through a hole is the portion on the part of the drawing plane where the hole is located. In other words, incorporating a design into a pattern does not change its alignment to the drawing plane, and does not apply a coordinate transformation to the design. Drawing a pattern collects the pieces of designs that show through all

the holes and draws the pieces where the holes lie on the drawing plane. The pattern is completely transparent outside the area defined by the array.

Each cell of a pattern occupies a 1 by 1 square. You can use `transform-region` to scale the pattern to a different cell size and shape, or to rotate the pattern so that the rectangular cells become diamond-shaped. Applying a coordinate transformation to a pattern does not affect the designs that make up the pattern. It only changes the position, size, and shape of the cells' holes, allowing different portions of the designs in the cells to show through. Consequently, applying `make-rectangular-tile` to a pattern of nonuniform designs can produce a different appearance in each tile. The pattern cells' holes are tiled, but the designs in the cells are not tiled and a different portion of each of those designs shows through in each tile.

<b>make-rgb-color</b>	<i>Function</i>								
Summary	Returns a member of class <code>&lt;color&gt;</code> .								
Signature	<code>make-rgb-color red green blue #key opacity =&gt; color</code>								
Arguments	<table> <tr> <td><i>red</i></td><td>An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code>.</td></tr> <tr> <td><i>green</i></td><td>An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code>.</td></tr> <tr> <td><i>blue</i></td><td>An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code>.</td></tr> <tr> <td><i>opacity</i></td><td>An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code>. Default value: 1.0.</td></tr> </table>	<i>red</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .	<i>green</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .	<i>blue</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .	<i>opacity</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> . Default value: 1.0.
<i>red</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .								
<i>green</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .								
<i>blue</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> .								
<i>opacity</i>	An instance of type <code>limited(&lt;real&gt;, min: 0, max: 1)</code> . Default value: 1.0.								
Values	<code>color</code> An instance of type <code>&lt;color&gt;</code> .								

Description	Returns a member of class <color>. The <i>red</i> , <i>green</i> , and <i>blue</i> arguments are real numbers between 0 and 1 (inclusive) that specify the values of the corresponding color components.  When all three color components are 1, the resulting color is white. When all three color components are 0, the resulting color is black.
See also	<code>make-gray-color</code> , page 144  <code>make-ihs-color</code> , page 145

## make-stencil *Function*

Summary	Returns a pattern design generated from a two-dimensional array.	
Signature	<code>make-stencil array =&gt; stencil</code>	
Arguments	<code>array</code>	An instance of type <array>.
Values	<code>stencil</code>	An instance of type <stencil>.
Description	Returns a pattern design that has ( <code>array-dimension array 0</code> ) cells in the vertical direction and ( <code>array-dimension array 1</code> ) cells in the horizontal direction. <code>array</code> must be a two-dimensional array of real numbers between 0 and 1 (inclusive) that represent opacities. The design in cell <i>i</i> , <i>j</i> of the resulting pattern is the value of ( <code>make-opacity (aref array i j)</code> ).	

## make-text-style *Function*

Summary	Returns an instance of <text-style>.
Signature	<code>make-text-style family weight slant size #key underline? strikeout? =&gt; text-style</code>

Arguments	<i>family</i>	An instance of type <code>one-of(#"fix", #"<b>serif</b>", #"<b>sans-serif</b>", #f).</code>
	<i>weight</i>	An instance of type <code>one-of(#"normal", #"<b>condensed</b>", #"<b>thin</b>", #"<b>extra-light</b>", #"<b>light</b>", #"<b>medium</b>", #"<b>demibold</b>", #"<b>bold</b>", #"<b>extra-bold</b>", #"<b>black</b>", #f).</code>
	<i>slant</i>	An instance of type <code>one-of(#"<b>roman</b>", #"<b>italic</b>", #"<b>oblique</b>", #f).</code>
	<i>size</i>	An instance of <code>&lt;integer&gt;</code> , or an instance of type <code>one-of(#"<b>normal</b>", #"<b>tiny</b>", #"<b>very-small</b>", #"<b>small</b>", #"<b>large</b>", #"<b>very-large:</b>", #"<b>huge</b>", #"<b>smaller</b>", #"<b>larger</b>", #f).</code>
	<i>underline?</i>	An instance of type <code>&lt;boolean&gt;</code> .
	<i>strikeout?</i>	An instance of type <code>&lt;boolean&gt;</code> .
Values	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Description	Returns an instance of <code>&lt;text-style&gt;</code> .  Text style objects have components for family, face, and size. Not all of these attributes need be supplied for a given text style object. Text styles can be merged in much the same way as pathnames are merged; unspecified components in the style object (that is, components that have <code>#f</code> in them) may be filled in by the components of a default style object. A text style object is called <i>fully specified</i> if none of its components is <code>#f</code> , and the size component is not a relative size (that is, neither <code>#"<b>smaller</b>"</code> nor <code>#"<b>larger</b>"</code> ).  If size is an integer, it represents the size of the font in printer's points.  Implementations are permitted to extend legal values for family, face, and size.	
See also	<code>\$solid-pen</code> , page 160.	

## merge-text-styles *Generic function*

Summary	Merges two text styles and returns a new text style that is the same as the first, except that unspecified components in are filled in from the second.	
Signature	<code>merge-text-styles <i>text-style default-style</i> =&gt; <i>text-style</i></code>	
Arguments	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
	<i>default-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Values	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Description	<p>Merges the text styles <i>text-style</i> with <i>default-style</i>, that is, returns a new text style that is the same as <i>text-style</i>, except that unspecified components in <i>text-style</i> are filled in from <i>default-style</i>. For convenience, the two arguments may be also be style specs. Note that <i>default-style</i> must be a <i>fully specified</i> text style.</p> <p>When merging the sizes of two text styles, if the size from the first style is a relative size, the resulting size is either the next smaller or next larger size than is specified by <i>default-style</i>. The ordering of sizes, from smallest to largest, is <code>#"tiny"</code>, <code>#"very-small"</code>, <code>#"small"</code>, <code>#"normal"</code>, <code>#"large"</code>, <code>#"very-large"</code>, and <code>#"huge"</code>.</p>	
See also	<code>default-background-setter</code> , page 127.	

## <palette> *Abstract instantiable class*

Summary	The protocol class for color palettes.
Superclasses	<code>&lt;object&gt;</code>
Init-keywords	None.

Description	The protocol class for color palettes.
Operations	<code>find-frame-manager frame-manager-palette-setter</code> <code>frame-palette-setter make make-frame-manager</code>
See also	<code>palette?</code> , page 151

## **palette?** *Generic function*

Summary	Returns #t if an object is a palette.	
Signature	<code>palette? object =&gt; boolean</code>	
Arguments	<code>object</code>	An instance of type <object>.
Values	<code>boolean</code>	An instance of type <boolean>.
Description	Returns #t if the object <i>object</i> is a palette. A palette is a color map that maps 16 bit colors into a, for example, 8 bit display.	
See also	<code>&lt;palette&gt;</code> , page 150	

## **<palette-full>** *Sealed concrete class*

Summary	The class for errors that are signalled when a color palette is full.	
Superclasses	<code>&lt;error&gt;</code>	
Init-keywords	<code>palette:</code>	
Description	The class for errors that are signalled when a color palette is full.	
See also	<code>&lt;palette&gt;</code> , page 150	

<b>\$parquet-stipple</b>		<i>Constant</i>
Summary	A stipple pattern for use in creating a patterned brush that looks like a parquet floor.	
Type	<code>&lt;array&gt;</code>	
Description	A stipple pattern for use in creating a patterned brush that looks like a parquet floor.	
See also	<code>brush-stipple</code> , page 115.	
<b>&lt;pattern&gt;</b>		<i>Sealed concrete class</i>
Summary	The class for patterns.	
Superclasses	<code>&lt;stencil&gt;</code>	
Init-keywords	<code>colors:</code>	An instance of type <code>limited(&lt;sequence&gt; of: &lt;color&gt;)</code> .
Description	The class for patterns. A pattern is a bounded rectangular arrangement of color, like a checkerboard. Drawing a pattern draws a different design in each rectangular cell of the pattern.	
Operations	The following operation is exported from the <code>DUIM-DCS</code> module.	
	<code>pattern?</code>	
See also	<code>&lt;stencil&gt;</code> , page 161 <code>make-pattern</code> , page 146	

## **pattern?** *Generic function*

Summary	Returns #t if its argument is a pattern.	
Signature	<b>pattern? object =&gt; boolean</b>	
Arguments	<i>object</i>	An instance of type <object>.
Values	<i>boolean</i>	An instance of type <boolean>.
Description	Returns #t if <i>object</i> is a pattern.	
See also	<a href="#">make-pattern</a> , page 146	

## **<pen>** *Abstract instantiable class*

Summary	The protocol class for pens.	
Superclasses	<object>	
Init-keywords	<b>width:</b>	An instance of type <integer>. Default value: 1.
	<b>units:</b>	An instance of type one-of(#"normal", #"point", #"device"). Default value: #"normal".
	<b>dashes:</b>	An instance of type union(<boolean>, <sequence>). Default value: #f.
	<b>joint-shape:</b>	An instance of type one-of(#"miter", #"bevel", #"round", #"none"). Default value: #"miter".
	<b>cap-shape:</b>	An instance of type one-of(#"butt", #"square", #"round", #"no-end-point"). Default value: #"butt".

Description	The protocol class for pens. A pen imparts ink to a medium.
Operations	<p>The following operations are exported from the <code>DUIM-DCs</code> module.</p> <pre>= pen? pen-cap-shape pen-dashes pen-joint-shape pen-units pen-width</pre>
See also	<p><code>&lt;ink&gt;</code>, page 138</p> <p><code>make</code>, page 140</p> <p><code>pen?</code>, page 154</p> <p><code>pen-cap-shape</code>, page 155</p> <p><code>pen-dashes</code>, page 155</p> <p><code>pen-joint-shape</code>, page 156</p> <p><code>pen-units</code>, page 157</p> <p><code>pen-width</code>, page 158</p>

<b>pen?</b>		<i>Generic function</i>
Summary		Returns <code>#t</code> if its argument is a pen.
Signature		<code>pen? object =&gt; boolean</code>
Arguments	<i>object</i>	An instance of type <code>&lt;object&gt;</code> .
Values	<i>boolean</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description		Returns <code>#t</code> if <i>object</i> is a pen, otherwise returns <code>#f</code> .
See also		<p><code>&lt;pen&gt;</code>, page 153</p> <p><code>pen-cap-shape</code>, page 155</p> <p><code>pen-dashes</code>, page 155</p>

`pen-joint-shape`, page 156

`pen-units`, page 157

`pen-width`, page 158

## pen-cap-shape

### *Generic function*

Summary      Returns the shape of the end of a line or an arc drawn by the pen.

Signature     `pen-cap-shape pen => value`

Arguments    `pen`                  An instance of type `<pen>`.

Values        `value`                  An instance of type `one-of(#"butt", "#square", "#round", "#no-end-point")`.

Description     Returns the shape of the end of a line or an arc drawn by `pen`.

See also      `make-contrasting-dash-patterns`, page 143

`<pen>`, page 153

`pen?`, page 154

`pen-dashes`, page 155

`pen-joint-shape`, page 156

`pen-units`, page 157

`pen-width`, page 158

## pen-dashes

### *Generic function*

Summary     Returns `#t` if the lines drawn by a pen are dashed.

Signature    `pen-dashes pen => value`

Arguments	<i>pen</i>	An instance of type <code>&lt;pen&gt;</code> .
Values	<i>value</i>	An instance of type <code>type-union(&lt;boolean&gt;, &lt;sequence&gt;)</code> .
Description	Returns <code>#t</code> if the lines drawn by <i>pen</i> are dashed. The sequence is a vector of integers indicating the pattern of dashes. There must be an even number of integers. The odd elements in the list indicate the length of the inked dashes and the even elements indicate the length of the gaps between dashes.	
See also	<a href="#">`&lt;pen&gt;`</a> , page 153 <a href="#">`pen?`</a> , page 154 <a href="#">`pen-cap-shape`</a> , page 155 <a href="#">`pen-joint-shape`</a> , page 156 <a href="#">`pen-units`</a> , page 157 <a href="#">`pen-width`</a> , page 158	

**pen-joint-shape***Generic function*

Summary	Returns the shape of the joints between line segments of a closed, unfilled figure.	
Signature	<code>pen-joint-shape pen =&gt; value</code>	
Arguments	<i>pen</i>	An instance of type <code>&lt;pen&gt;</code> .
Values	<i>value</i>	An instance of type <code>one-of(#"miter", "#"bevel", "#"round", "#"none")</code> .
Description	Returns the shape of the joints between line segments of a closed, unfilled figure drawn by <i>pen</i> .	

See also [make-contrasting-dash-patterns](#), page 143  
[<pen>](#), page 153  
[pen?](#), page 154  
[pen-cap-shape](#), page 155  
[pen-dashes](#), page 155  
[pen-units](#), page 157  
[pen-width](#), page 158

## **pen-units** *Generic function*

Summary Returns the units in which the pen width is specified.

Signature **pen-units** *pen* => *value*

Arguments *pen* An instance of type [<pen>](#).

Values *value* An instance of type `one-of(#"normal",  
#"point", #"device")`.

Description Returns the units in which the pen width is specified. They may be normal, points, or device-dependent. A width of `#"normal"` is a comfortably visible thin line.

See also [make-contrasting-dash-patterns](#), page 143  
[<pen>](#), page 153  
[pen?](#), page 154  
[pen-cap-shape](#), page 155  
[pen-dashes](#), page 155  
[pen-joint-shape](#), page 156  
[pen-width](#), page 158

	<i>Generic function</i>
<b>pen-width</b>	
Summary	Returns the pen-width, that is how wide a stroke the pen draws, of its argument.
Signature	<b>pen-width</b> <i>pen</i> => <i>width</i>
Arguments	<i>pen</i> An instance of type <code>&lt;pen&gt;</code> .
Values	<i>width</i> An instance of type <code>&lt;pen-width&gt;</code> . The units that specify the width of the pen may be <code>#"normal"</code> , <code>#"points"</code> , or <code>#"device"</code> .
Description	Returns the pen width, that is how wide a stroke the pen draws, of <i>pen</i> . A width of <code>#"normal"</code> is a comfortably visible thin line.
See also	<a href="#">make-contrasting-dash-patterns</a> , page 143 <a href="#">&lt;pen&gt;</a> , page 153 <a href="#">pen?</a> , page 154 <a href="#">pen-cap-shape</a> , page 155 <a href="#">pen-dashes</a> , page 155 <a href="#">pen-joint-shape</a> , page 156 <a href="#">pen-units</a> , page 157

	<i>Generic function</i>
<b>read-image</b>	
Summary	Reads an image.
Signature	<b>read-image</b> <i>resource-id</i> #key <i>image-type</i> : <i>image-type</i> #all-keys => <i>image</i>
Arguments	<i>locator</i> An instance of type <code>type-union(&lt;string&gt;, &lt;locator&gt;)</code> .

	<i>image-type</i>	On Windows, an instance of type <code>one-of(#"bitmap", #"icon")</code> .
Values	<i>image</i>	An instance of type <code>&lt;image&gt;</code> .
Description		Reads an image from the location <i>resource-id</i> . This function calls <code>read-image-as</code> .
See also		<code>read-image-as</code> , page 159.

## **read-image-as** *Generic function*

Summary	Reads an image.	
Signature	<code>read-image-as class locator image-type #key #all-keys =&gt; image</code>	
Arguments	<i>class</i>	An instance of type <code>&lt;object&gt;</code> .
	<i>locator</i>	An instance of type <code>&lt;string&gt;</code> .
	<i>image-type</i>	On Windows, <code>#"bitmap"</code> or <code>#"icon"</code> .
Values	<i>image</i>	An instance of type <code>&lt;image&gt;</code> .
Description	Reads the image in the location pointed to be <i>locator</i> , as an instance of a particular class. This function is called by <code>read-image</code> .  The <i>class</i> represents the class that the image is read as an instance of.	
See also	<code>read-image</code> , page 158	

## **\$red** *Constant*

Summary	The usual definition of the color red.
---------	--

Type	<code>&lt;color&gt;</code>
Description	The usual definition of the color red.
See also	See the class <code>\$blue</code> , page 106.

## **remove-colors** *Generic function*

Summary	Removes one or more colors from a palette and returns the updated palette.				
Signature	<code>remove-colors palette #rest colors =&gt; palette</code>				
Arguments	<table><tr><td><i>palette</i></td><td>An instance of type <code>&lt;palette&gt;</code>.</td></tr><tr><td><i>colors</i></td><td>Instances of type <code>&lt;color&gt;</code>.</td></tr></table>	<i>palette</i>	An instance of type <code>&lt;palette&gt;</code> .	<i>colors</i>	Instances of type <code>&lt;color&gt;</code> .
<i>palette</i>	An instance of type <code>&lt;palette&gt;</code> .				
<i>colors</i>	Instances of type <code>&lt;color&gt;</code> .				
Values	<i>palette</i>				
Description	Removes <i>colors</i> from <i>palette</i> and returns the updated palette.				

## **\$solid-pen** *Constant*

Summary	A pen that draws a solid line.
Type	<code>&lt;pen&gt;</code>
Description	A pen that draws a solid line. The width of the line is <code>1</code> , and <code>dashes:</code> is <code>#f</code> .
See also	See the class <code>&lt;pen&gt;</code> , page 153 and the constants <code>make</code> , page 140, <code>\$dash-dot-pen</code> , page 126, and <code>\$dotted-pen</code> , page 131.

<stencil>		<i>Sealed concrete class</i>
Summary	The class for stencils.	
Superclasses	<image>	
Init-keywords	<b>array:</b> An instance of type <array>. Required. <b>transform:</b> An instance of type <transform>. Default value: #f.	
Description	The class for stencils. A <i>stencil</i> is a special kind of pattern that contains only opacities.	
Operations	The following operations are exported from the <b>DUIM-DCs</b> module.  <b>image-height</b> <b>image-width</b> <b>stencil?</b>  The following operation is exported from the <b>DUIM-Geometry</b> module.  <b>box-edges</b>	
See also	<a href="#">&lt;image&gt;, page 136</a> <a href="#">make-pattern, page 146</a> <a href="#">stencil?, page 161</a>	

<b>stencil?</b>		<i>Generic function</i>
Summary	Returns #t if its argument is a stencil.	
Signature	<b>stencil? object =&gt; boolean</b>	
Arguments	<b>object</b> An instance of type <object>.	
Values	<b>boolean</b> An instance of type <boolean>.	

Description      Returns `#t` if its argument is a stencil.

See also      `make-pattern`, page 146.

`<stencil>`, page 161

## `<text-style>` *Abstract instantiable class*

Summary      The protocol class for text styles.

Superclasses    `<object>`

Init-keywords    `family:`      An instance of type `one-of(#"fix",  
#"serif", #"sans-serif", #f)`. Default value: `#f`.

`weight:`      An instance of type `one-of(#"normal",  
#"condensed", #"thin", #"extra-light",  
#"light", #"medium", #"demibold",  
#"bold", #"extra-bold", #"black", #f)`.

`slant:`      An instance of type `one-of(#"roman",  
#"italic", #"oblique", #f)`.

`size:`      An instance of `<integer>`, or an instance of type `one-of(#"normal", #"tiny",  
#"very-small", #"small", #"large",  
#"very-large:", #"huge", #"smaller",  
#"larger", #f)`. Default value: `#f`.

`underline?:`      An instance of type `<boolean>`. Default value: `#f`.

`strikeout?:`      An instance of type `<boolean>`. Default value: `#f`.

Description      The protocol class for text styles. When specifying a particular appearance for rendered characters, there is a tension between portability and access to specific font for a display

device. DUIM provides a portable mechanism for describing the desired *text style* in abstract terms. Each port defines a mapping between these abstract style specifications and particular device-specific fonts. In this way, an application programmer can specify the desired text style in abstract terms secure in the knowledge that an appropriate device font will be selected at run time. However, some applications may require direct access to particular device fonts. The text style mechanism supports specifying device fonts by name, allowing the programmer to sacrifice portability for control.

If `size:` is specified as an integer, then it represents the font size in printer's points.

Operations	<p>The following operations are exported from the <b>DUIM-DCs</b> module.</p> <pre>= fully-merged-text-style? merge-text-styles text-style? text-style-components text-style-family text-style-size text-style-slant text-style-strikeout? text-style-underline? text-style-weight</pre>
The following operations are exported from the <b>DUIM-Sheets</b> module.	<pre>fixed-width-font? font-ascent font-descent font-height font-metrics font-width text-style-mapping text-style-mapping-exists? text-style-mapping-setter</pre>
See also	<p><code>text-style?</code>, page 164</p> <p><code>text-style-components</code>, page 164</p> <p><code>text-style-family</code>, page 165</p> <p><code>text-style-size</code>, page 166</p> <p><code>text-style-slant</code>, page 167</p> <p><code>text-style-strikeout?</code>, page 168</p> <p><code>text-style-underline?</code>, page 168</p>

`text-style-weight`, page 169

## **text-style?** *Generic function*

Summary	Returns <code>#t</code> if its argument is a text-style.
Signature	<code>text-style? object =&gt; text-style?</code>
Arguments	<code>object</code> An instance of type <code>&lt;object&gt;</code> .
Values	<code>text-style?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns <code>#t</code> if its argument is a text-style.
See also	<code>&lt;text-style&gt;</code> , page 162 <code>text-style-components</code> , page 164 <code>text-style-family</code> , page 165 <code>text-style-size</code> , page 166 <code>text-style-slant</code> , page 167 <code>text-style-strikeout?</code> , page 168 <code>text-style-underline?</code> , page 168 <code>text-style-weight</code> , page 169

## **text-style-components** *Generic function*

Summary	Returns the components of a text style as the values family, face, slant, size, underline and strikeout.
Signature	<code>text-style-components text-style =&gt; family weight slant size underline? strikeout?</code>
Arguments	<code>text-style</code> An instance of type <code>&lt;text-style&gt;</code> .

Values	<i>family</i>	An instance of type <code>one-of(#"fix", #"<b>serif</b>", #"<b>sans-serif</b>", #f).</code>
	<i>weight</i>	An instance of type <code>one-of(#"normal", #"<b>condensed</b>", #"<b>thin</b>", #"<b>extra-light</b>", #"<b>light</b>", #"<b>medium</b>", #"<b>demibold</b>", #"<b>bold</b>", #"<b>extra-bold</b>", #"<b>black</b>", #f).</code>
	<i>slant</i>	An instance of type <code>one-of(#"<b>roman</b>", #"<b>italic</b>", #"<b>oblique</b>", #f).</code>
	<i>size</i>	An instance of <code>&lt;integer&gt;</code> , or an instance of type <code>one-of(#"<b>normal</b>", #"<b>tiny</b>", #"<b>very-small</b>", #"<b>small</b>", #"<b>large</b>", #"<b>very-large:</b>", #"<b>huge</b>", #"<b>smaller</b>", #"<b>larger</b>", #f)</code> . Default value: #f.
	<i>underline?</i>	An instance of type <code>&lt;boolean&gt;</code> .
	<i>strikeout?</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns the components of the text style <i>text-style</i> as the values family, face, slant, size, underline and strikeout.	
See also	<a href="#">&lt;text-style&gt;</a> , page 162 <a href="#">text-style?</a> , page 164 <a href="#">text-style-family</a> , page 165 <a href="#">text-style-size</a> , page 166 <a href="#">text-style-slant</a> , page 167 <a href="#">text-style-strikeout?</a> , page 168 <a href="#">text-style-underline?</a> , page 168 <a href="#">text-style-weight</a> , page 169	

**text-style-family***Generic function*

Summary      Returns the family component of the specified text style.

Signature	<code>text-style-family</code> <i>text-style</i> => <i>family</i>	
Arguments	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Values	<i>family</i>	An instance of type <code>one-of(#"fix", "#serif", "#sans-serif", #f)</code> .
Description	Returns the family component of the specified text style.	
See also	<code>&lt;text-style&gt;</code> , page 162 <code>text-style?</code> , page 164 <code>text-style-components</code> , page 164 <code>text-style-size</code> , page 166 <code>text-style-slant</code> , page 167 <code>text-style-strikeout?</code> , page 168 <code>text-style-underline?</code> , page 168 <code>text-style-weight</code> , page 169	

## **text-style-size** *Generic function*

Summary	Returns the style component of the specified text style.	
Signature	<code>text-style-size</code> <i>text-style</i> => <i>size</i>	
Arguments	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Values	<i>size</i>	An instance of <code>&lt;integer&gt;</code> , or an instance of type <code>one-of(#"normal", #"tiny", "#very-small", #"small", #"large", "#very-large:", #"huge", #"smaller", #"larger", #f)</code> . Default value: <code>#f</code> .
Description	Returns the style component of the specified text style.	

## See also

[`<text-style>`](#), page 162  
[`text-style?`](#), page 164  
[`text-style-components`](#), page 164  
[`text-style-family`](#), page 165  
[`text-style-slant`](#), page 167  
[`text-style-strikeout?`](#), page 168  
[`text-style-underline?`](#), page 168  
[`text-style-weight`](#), page 169

**`text-style-slant`***Generic function*

## Summary

Returns the slant component of the specified text style.

## Signature

`text-style-slant text-style => slant`

## Arguments

`text-style` An instance of type [`<text-style>`](#).

## Values

`slant` An instance of type `one-of(#"roman", "#"italic", "#"oblique", #f)`.

## Description

Returns the slant component of the specified text style.

## See also

[`<text-style>`](#), page 162  
[`text-style?`](#), page 164  
[`text-style-components`](#), page 164  
[`text-style-family`](#), page 165  
[`text-style-size`](#), page 166  
[`text-style-strikeout?`](#), page 168  
[`text-style-underline?`](#), page 168  
[`text-style-weight`](#), page 169

## **text-style-strikeout?** *Generic function*

Summary	Returns #t if the text style includes a line through it, striking it out.
Signature	<code>text-style-strikeout? text-style =&gt; strikeout?</code>
Arguments	<code>text-style</code> An instance of type <code>&lt;text-style&gt;</code> .
Values	<code>strikeout?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns #t if the text style includes a line through it, striking it out.
See also	<code>&lt;text-style&gt;</code> , page 162 <code>text-style?</code> , page 164 <code>text-style-components</code> , page 164 <code>text-style-family</code> , page 165 <code>text-style-size</code> , page 166 <code>text-style-slant</code> , page 167 <code>text-style-underline?</code> , page 168 <code>text-style-weight</code> , page 169

## **text-style-underline?** *Generic function*

Summary	Returns #t if the text style is underlined.
Signature	<code>text-style-underline? text-style =&gt; underline?</code>
Arguments	<code>text-style</code> An instance of type <code>&lt;text-style&gt;</code> .
Values	<code>underline?</code> An instance of type <code>&lt;boolean&gt;</code> .

Description	Returns <code>#t</code> if the text style is underlined.
See also	<code>&lt;text-style&gt;</code> , page 162 <code>text-style?</code> , page 164 <code>text-style-components</code> , page 164 <code>text-style-family</code> , page 165 <code>text-style-size</code> , page 166 <code>text-style-slant</code> , page 167 <code>text-style-strikeout?</code> , page 168 <code>text-style-weight</code> , page 169

## text-style-weight

## *Generic function*

Summary	Returns the weight component of the specified text style.	
Signature	<code>text-style-weight <i>text-style</i> =&gt; <i>weight</i></code>	
Arguments	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Values	<i>weight</i>	An instance of type <code>one-of(#"normal", #"condensed", #"thin", #"extra-light", #"light", #"medium", #"demibold", #"bold", #"extra-bold", #"black", #f)</code> .
Description	Returns the weight component of the text style.	
See also	<code>&lt;text-style&gt;</code> , page 162 <code>text-style?</code> , page 164 <code>text-style-components</code> , page 164 <code>text-style-family</code> , page 165 <code>text-style-size</code> , page 166	

**text-style-slant**, page 167  
**text-style-strikeout?**, page 168  
**text-style-underline?**, page 168

**\$tiles-stipple** *Constant*

Summary     A stipple pattern for use in creating a patterned brush with lines and spaces suggesting tiles

Type        **<array>**

Description   A stipple pattern for use in creating a patterned brush with lines and spaces suggesting tiles

See also    **brush-stipple**, page 115.

**\$vertical-hatch** *Constant*

Summary     A stipple pattern for use in creating a patterned brush with alternating vertical columns of lines and spaces.

Type        **<array>**

Description   A stipple pattern for use in creating a patterned brush with alternating vertical columns of lines and spaces.

See also    **brush-stipple**, page 115.

**\$white** *Constant*

Summary     The usual definition of white.

Type        **<color>**

Description      The usual definition of white. In the `rgb` color model, its value is `111`.

See also      `<color>`, page 117

## **write-image**

*Generic function*

Summary      Writes out a copy of an image to disk (or other designated medium).

Signature      `write-image image locator => ()`

Arguments      `image`      An instance of type `<image>`.  
                   `locator`      An instance of type `<string>`.

Values      None

Description      Writes out a copy of `image` to the designated medium `locator`.

## **\$xor-brush**

*Constant*

Summary      A standard brush with the drawing property of `$boole-xor`.

Type      `<brush>`

Description      A standard brush with the drawing property of `$boole-xor`.

## **\$yellow**

*Constant*

Summary      The usual definition of the color yellow.

Type      `<color>`

Description      The usual definition of the color yellow.

See also      `<color>`, page 117

# 5

---

# DUIM-Sheets Library

## 5.1 Overview

The elements that comprise a Graphical User Interface (GUI) are arranged in a hierarchical ordering of object classes. At the top level of the DUIM hierarchy there are three main classes, `<sheet>`, `<gadget>`, and `<frame>`, all of which are subclasses of `<object>`.

Sheets are the most basic visual GUI element, and can be any unique part of a window: either a control such as a gadget or pane, or a layout.

- Sheets have a visual presence: size, drawing context and so on.
- The essential component of a sheet is its region; the area of the screen that the sheet occupies.
- In practice sheets always also have a transform that maps the coordinate system of the sheet's region to the coordinate system of its parent, because in practice all sheets maintain a pointer to a parent sheet.
- Sheets can be output-only (labels, for example), input-output (most gadgets are like this) or even, in principle, input-only (for instance, you may need to provide some kind of simple drag'n'drop target).

Most of the sheet classes that you need to use on a day to day basis are exposed in the DUIM-Gadgets and DUIM-Layouts libraries. The DUIM-Sheets library contains the basic building blocks to implement these classes, as well as providing the necessary functionality for you to create and manipulate your own classes of sheet. In addition, DUIM-Sheets defines a portable model for handling events. These event handling routines are used by the DUIM-Frames, DUIM-Gadgets, and DUIM-Layouts libraries without the need for any special action on your part. However, if you need to define your own sheet classes, you will also need to handle events occurring within those classes.

The DUIM-Sheets library contains a single module, `duim-sheets`, from which all the interfaces described in this chapter are exposed. Section 5.3 on page 183 contains complete reference entries for each exposed interface.

A sheet is the basic unit in a DUIM window. Inside any window, sheets are nested in a parent-child hierarchy. All sheets have the following attributes:

- `sheet-region`, expressed in the sheet's own coordinate system.
- `sheet-transform`, which maps the sheet's coordinate system to the coordinate system of its parent.
- `sheet-parent`, which is `#f` if the sheet has no parent.
- `sheet-mapped?`, which tells if the sheet is visible on a display, ignoring issues of occluding windows.

The `sheet-transform` is an instance of a concrete subclass of `<transform>`. The `sheet-region` can be an instance of any concrete subclass of `<region>`, but is usually represented by the region class `<bounding-box>`.

Some sheets (menu bars, button boxes, or tool bars, for instance) also have single or multiple children, in which case they have additional attributes:

- A `sheet-children` slot. This is a sequence of sheets. Each sheet in the sequence is a child of the current sheet.
- Methods to add, remove, and replace a child.
- Methods to map over children.

The functions that maintain the sheet's region and transform are part of the `sheet-geometry` protocol. Functions that maintain a sheet's parent and children are part of the `sheet-genealogy` protocol. Note that the sheet geometry and genealogy protocols are independent. Adding a child to a sheet that is larger than its parent does not cause the parent's region to grow. Shrinking the region of a parent does not cause the children to shrink. You must maintain the region yourself, either by explicitly setting the sheet's region and transform, or by using the layout facilities (`compose-space` and `allocate-space`).

As a convenience, there are some *glue* functions that mediate between geometry and layout: `set-sheet-position`, `set-sheet-size`, and `set-sheet-edges`.

Some classes of sheet can receive input. These have:

- A `sheet-event-queue` slot.
- Methods for `<handle-event>`.

Sheets that can be repainted have methods for `handle-repaint`.

Sheets that can do output, have a `sheet-medium` slot.

Some sheets act as *controls* such as push buttons, scroll bars, and sliders. These are represented by the `<gadget>` class and its subclasses.

Other sheets act as layout controls, which allow you to specify how the elements in a sheet are laid out, whether they are placed vertically or horizontally, whether they are left, right, or center-aligned, and so on. These are represented by the `<layout>` class and its subclasses, and are described in Chapter 7, “DUIM-Layouts Library”.

A sheet can be associated with a `<display>`, which is an object that represents a single display (or screen) on some display server.

A display (and all the sheets attached to the display) is associated with a `<port>` that is a connection to a display server. The port manages:

- a primary input device, such as a keyboard.
- a pointing device, such as a mouse.
- an event processor that *dispatches* events to the appropriate sheet.

There is a protocol for using the Windows clipboard. 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.

## 5.2 The class hierarchy for DUIM-Sheets

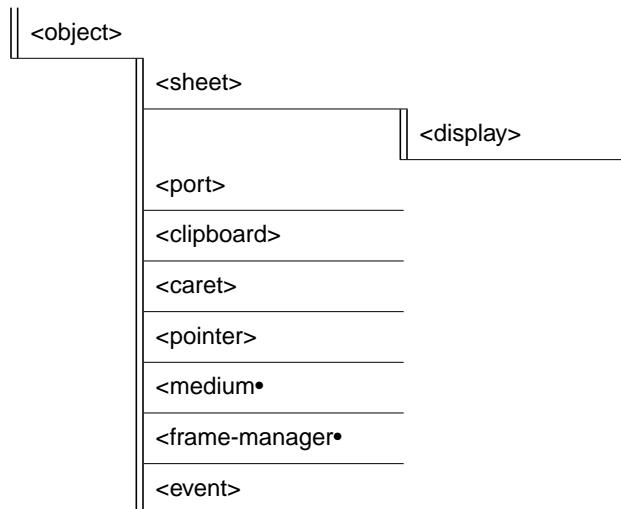
This section presents an overview of the available classes exposed by the DUIM-Sheets library, and describes the class hierarchy present.

### 5.2.1 The base classes in the DUIM-Sheets library

The base classes for the majority of subclasses exposed from the DUIM-Sheets library are `<sheet>` and `<event>`, although a number of additional subclasses of `<object>` are also exposed.

The base classes exposed by the DUIM-Sheets library are shown in Table 5.1. Only `<sheet>`, and `<event>` have any subclasses defined. An `<event>` is an object representing some sort of event. See Section 5.2.2 on page 179 for details of the subclasses of `<event>`.

**Table 5.1** Overall class hierarchy for the DUIM-Sheets library



<code>&lt;sheet&gt;</code>	As already mentioned, a sheet is the basic unit of window applications, and they can be nested in a parent-child hierarchy. A subclass of sheet is provided — <code>&lt;display&gt;</code> — which is an object that represents a single display (or screen) on a display server. All sheets can be attached to a display.
<code>&lt;port&gt;</code>	A port is a connection to a display server. A display, together with all the sheets attached to it, is associated with a port, which manages a primary input device, such as a keyboard, a pointing device, such as a mouse, and an event processor that dispatches events to the appropriate sheet.

**<clipboard>** This class is used as a clipboard that can be used to hold information temporarily while it is transferred from one sheet to another, or between applications. Clipboards provide support for the standard **Cut**, **Copy**, and **Paste** commands common in most applications.

**<caret> and <pointer>**

These two classes form an interface between the keyboard and the display, and the pointing device and the display, respectively.

The **<caret>** represents the position on screen that characters typed on the keyboard will be placed. This is often a position in a document.

The **<pointer>** represents the position of the pointing device on the screen, and thus shows the area that will be affected by any events generated with the pointing device, such as pressing or clicking one of the buttons on the device.

**<pointer-drag-event>**

The class of events where the pointer for the pointing device attached to the computer is moving, and one of the buttons on the pointing device is pressed down as well. The effects of this event are rather like a combination of the **<button-press-event>** and **<pointer-motion-event>** classes. For more information about these and other pointer event classes, see Section 5.2.3 on page 181.

**<pointer-enter-event>**

This event is used to describe the event where the pointer for the pointing device enters a specified area of the screen, such as a sheet. For more information about these and other pointer event classes, see Section 5.2.3 on page 181.

**<medium>**

A medium represents a destination for drawn or written output. It has several items associated with it, such as a drawing plane, foreground and background colors, and default line and text styles.

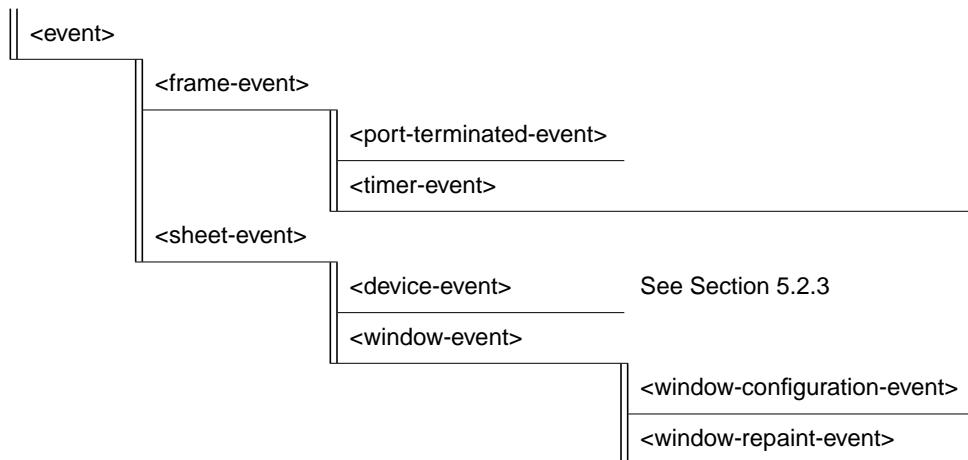
**<frame-manager>**

A frame manager represents the “look and feel” of a frame. This controls standard interface elements for the platform you are delivering on, such as the appearance and behavior of title bars, borders, menu commands and scroll bars. Unless you are developing for more than one platform, you do not need to be too concerned with frame managers, since you will only be using the default frame manager.

### 5.2.2 Subclasses of <event>

Table 5.2 shows the subclasses of the **<event>** class that are exposed by the DUIM-Sheets library.

**Table 5.2** Subclasses of the **<event>** class



The classes of event that are exposed by the DUIM-Sheets library fall into two categories:

- Events that occur in frames: subclasses of the `<frame-event>` class
- Events that occur in sheets: subclasses of the `<sheet-event>` class

Most subclasses of `<frame-event>` are exposed by the DUIM-Frames library. See Chapter 9, “DUIM-Frames Library”, for full details about these subclasses. However, two subclasses of `<frame-event>` are exposed by the DUIM-Sheets library:

`<port-terminated-event>`

This class represents the event of a port — a connection to a display — being terminated.

`<timer-event>` This is the class of any event that is timed.

Subclasses of `<sheet-event>` fall into two categories:

- Device events that occur to devices attached to the computer (typically the keyboard and the pointing device). These are described in Section 5.2.3 on page 181.
- Window events that occur in a window.

Events that occur in a window are subclasses of `<window-event>`. Two such events are supplied:

`<window-configuration-event>`

This event occurs whenever the configuration of sheets in a window needs to be recalculated. This may occur in property frames, for example, when clicking on the available tabs to display different pages of information.

Sometimes, dialog boxes have buttons that allow you to show or hide additional details, which are themselves displayed in an extra pane at the bottom or on the right hand side of the dialog. Clicking on such a button would also create a `<window-configuration-event>`, as the additional pane would need to be displayed or hidden, forcing a recalculation of the layout of the sheets in the frame.

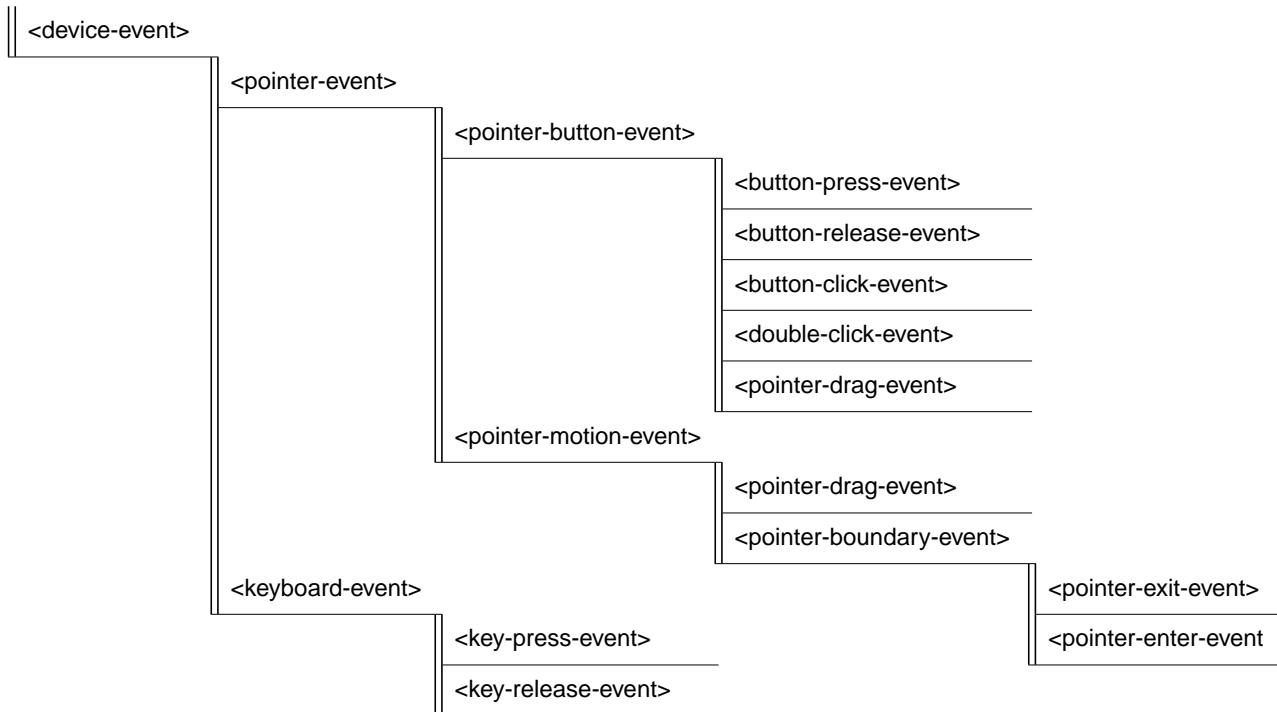
```
<window-repaint-event>
```

This event occurs whenever a region of a window needs to be repainted. This may occur when refreshing a chart or drawing in a frame.

### 5.2.3 Subclasses of <device-event>

Table 5.3 shows the subclasses of the <device-event> class that are exposed by the DUIM-Sheets library. Device events, broadly speaking, describe any event that can occur on a device connected to the computer.

**Table 5.3** Subclasses of the <device-event> class



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

Device events fall into two distinct categories:

- Keyboard events that occur on the keyboard attached to the computer: subclasses of `<keyboard-event>`
- Pointer events that occur on the pointing device attached to the computer: subclasses of `<pointer-event>`

There are two classes of keyboard event. The classes `<key-press-event>` and `<key-release-event>` describe the events that occur when any key on the keyboard is pressed or released, respectively.

There are three classes of pointer event, some of which provide a number of subclasses. Note that there are another two classes of pointer event that are immediate subclasses of `<object>`. These are described in Section 5.2.1 on page 176.

`<pointer-button-event>`

These events occur whenever there is any activity on one of the buttons on the pointing device. Several subclasses of this class are provided.

`<pointer-exit-event>`

This is an event that occurs when the pointer leaves a specified area such as a sheet.

`<pointer-motion-event>`

This class of events occur when the pointer is in motion. There is one subclass provided, `<pointer-boundary-event>`, for the specific case when the motion of the pointer causes the boundary of a sheet to be crossed.

**Note:** Unlike `<pointer-drag-event>`, no button needs to be pressed on the attached pointing device.

The subclasses provided for `<pointer-button-event>` are as follows:

`<button-press-event>`

This event occurs when any button on the pointing device is pressed down by the user. Note that this is distinct from `<button-click-event>`, described below.

`<button-release-event>`

This event occurs when any previously pressed button on the pointing device is released by the user.

`<button-click-event>`

This event occurs when any button on the pointing device is pressed down by the user and then released again within a certain time frame. An instance of this class is created if the creation of an instance of `<button-press-event>` is closely followed by the creation of an instance of `<button-release-event>`. The necessary time frame is dictated by the configuration of your computer. In Windows, for example, this time can be set using the Control Panel.

`<double-click-event>`

This event occurs when a button is clicked twice within a certain time frame. An instance of this class is created if the creation of an instance of `<button-click-event>` is closely followed by the creation of another instance of `<button-click-event>`. The necessary time frame is dictated by the configuration of your computer.

## 5.3 DUIM-Sheets Module

This section contains a complete reference of all the interfaces that are exported from the `duim-sheets` module.

=

*G.f. method*

Summary      Returns true if the specified gestures are the same.

Signature      = `gesture1 gesture2 => equal?`

Arguments      `gesture1`      An instance of type `<gesture>`.

`gesture2`      An instance of type `<gesture>`.

Values	<i>equal?</i>	An instance of type <boolean>.
Description	Returns true if <i>gesture1</i> and <i>gesture2</i> are the same.	
See also	<a href="#">gesture-spec-equal</a> , page 249	

**add-child** *Generic function*

Summary	Adds a child to the specified sheet.	
Signature	<code>add-child sheet child #key index =&gt; sheet</code>	
Arguments	<i>sheet</i>	An instance of type <sheet>.
	<i>child</i>	An instance of type <sheet>.
	<i>index</i>	An instance of type <code>false-or(&lt;integer&gt;)</code> .
Values	<i>sheet</i>	An instance of type <sheet>.
Description	Adds a child to <i>sheet</i> .	
See also	<a href="#">remove-child</a> , page 294 <a href="#">replace-child</a> , page 295	

**add-clipboard-data** *Generic function*

Summary	Adds data to a clipboard.	
Signature	<code>add-clipboard-data clipboard data =&gt; success?</code>	
Arguments	<i>clipboard</i>	An instance of <clipboard>.
	<i>data</i>	An instance of <object>.
Values	<i>success?</i>	An instance of <boolean>.

Description	This generic function adds <i>data</i> to <i>clipboard</i> . It returns #t if <i>data</i> was successfully added to the clipboard.
-------------	--

## **add-clipboard-data-as** *Generic function*

Summary	Coerces data to a particular type and then adds it to a clipboard.
---------	--

Signature	<b>add-clipboard-data</b> <i>type clipboard data</i> => <i>success?</i>
-----------	---

Arguments	<i>type</i> An instance of <code>type-union(&lt;symbol&gt;, &lt;type&gt;)</code> .
	<i>clipboard</i> An instance of <code>&lt;clipboard&gt;</code> .
	<i>data</i> An instance of <code>&lt;object&gt;</code> .

Values	<i>success?</i> An instance of <code>&lt;boolean&gt;</code> .
--------	---

Description	This generic function adds <i>data</i> to <i>clipboard</i> , first coercing it to <i>type</i> . The argument <i>type</i> is an instance of <code>type-union(&lt;symbol&gt;, &lt;type&gt;)</code> . It returns #t if <i>data</i> was successfully added to the clipboard.
-------------	--

## **\$alt-key** *Constant*

Summary	A constant that represents the ALT key on the keyboard.
---------	---

Type	<code>&lt;integer&gt;</code>
------	------------------------------

Value	<code>\$meta-key</code>
-------	-------------------------

Description	A constant that represents the ALT key on the keyboard. This is set to the same value as the META key, to deal with the case where the META key is not present on the keyboard.
-------------	---

See also

- \$control-key, page 211
- \$hyper-key, page 254
- \$meta-key, page 271
- modifier-key-index, page 273
- modifier-key-index-name, page 273
- \$modifier-keys, page 274
- \$option-key, page 278
- \$shift-key, page 317
- \$super-key, page 318

**beep** *Generic function*

## Summary

### Signature

Arguments     *drawable*       An instance of type `type-union(<sheet>, <medium>)`.

## Values      None

## Description

**boundary-event-kind** *Generic function*

**Summary** Returns the kind of boundary event for the specified event.

Signature      **boundary-event-kind** *event* => *symbol*

Arguments      *event*      An instance of type `<event>`.

Values	<i>symbol</i>	An instance of type <code>one-of(#"ancestor", #"virtual", #"inferior", #"nonlinear", #"nonlinear-virtual", #f)</code> .
Description		Returns the kind of boundary event for <i>event</i> . These correspond to the detail members for X11 enter and exit events.
See also		<code>&lt;pointer-boundary-event&gt;</code> , page 279

## button-index *Inline function*

Summary	Returns the index for the specified pointer button.	
Signature	<code>button-index <i>button</i> =&gt; <i>index</i></code>	
Arguments	<i>button</i>	An instance of type <code>one-of(#"left", #"middle", #"right")</code> .
Values	<i>index</i>	An instance of type <code>&lt;integer&gt;</code> .
Description	Returns the index for <i>button</i> , a button on the pointer device connected to the computer (typically a mouse). The <i>index</i> returned is either 0, 1, or 2, for the left, middle, or right buttons, respectively.	
See also	<a href="#">button-index-name, page 187</a> <a href="#">\$pointer-buttons, page 281</a>	

## button-index-name *Function*

Summary	Returns the button on the pointer device represented by the specified index.
Signature	<code>button-index-name <i>index</i> =&gt; <i>button</i></code>

Arguments	<i>index</i>	An instance of type <integer>.
Values	<i>button</i>	An instance of type one-of(#"left", #"middle", #"right").
Description	Returns the button on the pointer device connected to the computer (typically a mouse) represented by <i>index</i> . The <i>index</i> is either 0, 1, or 2, these values corresponding to the left, middle, or right buttons, respectively.	
See also	<a href="#">button-index</a> , page 187 <a href="#">\$pointer-buttons</a> , page 281	

## <button-press-event> *Sealed instantiable class*

Summary	The class of events representing button presses.
Superclasses	<a href="#">&lt;pointer-button-event&gt;</a>
Init-keywords	None.
Description	The class of events representing button presses. A instance of this class is generated if a button press is detected, and a second button press is not detected within the allowed interval for a double-click event. Alternatively, if a double-click event has just been generated, then an instance of this class is generated when a subsequent button press is detected.
Operations	None.
See also	<a href="#">&lt;button-release-event&gt;</a> , page 189 <a href="#">&lt;double-click-event&gt;</a> , page 226

**<button-release-event>***Sealed instantiable class*

**Summary** The class of events representing button releases.

**Superclasses** `<pointer-button-event>`

**Init-keywords** None.

**Description** The class of events representing button releases. An instance of this class is generated if the mouse button is released after a period of being pressed, for example, at the end of a drag and drop maneuver.

**Operations** None.

**See also** `<button-press-event>`, page 188

**<caret>***Abstract instantiable class*

**Summary** The class of caret.

**Superclasses** `<object>`

**Init-keywords** `sheet:` An instance of type `false-or(<sheet>)`.

`x:` An instance of type `<integer>`. Default value: 0.

`y:` An instance of type `<integer>`. Default value: 0.

`width:` An instance of type `<integer>`. Default value: 0.

`height:` An instance of type `<integer>`. Default value: 0.

Description	<p>The class of carets, or text cursors. A cursor can actually be any instance of <code>&lt;symbol&gt;</code> or any instance of <code>&lt;image&gt;</code>.</p> <p>The <code>sheet:</code> init-keyword specifies the sheet that the caret is positioned in.</p> <p>The <code>x::</code>, <code>y::</code>, <code>width::</code>, and <code>height::</code> init-keywords define the position and size of the caret, with respect to the sheet that contains it. The position of the caret is measured from the top left of the sheet. All units are measured in pixels.</p>
Operations	<p>The following operations are exported from the <code>DUIM-Sheets</code> module.</p> <pre><code>caret-position caret-sheet caret-size caret-visible? caret-visible?-setter display port set-caret-position</code></pre>
See also	<p><code>caret-position</code>, page 190</p> <p><code>caret-sheet</code>, page 191</p> <p><code>caret-size</code>, page 191</p> <p><code>caret-visible?</code>, page 192</p> <p><code>&lt;cursor&gt;</code>, page 211</p>

## `caret-position` *Generic function*

Summary	Returns the position of the specified caret.	
Signature	<code>cursor-position caret =&gt; xy</code>	
Arguments	<code>caret</code>	An instance of type <code>&lt;caret&gt;</code> .
Values	<code>x</code>	An instance of type <code>&lt;integer&gt;</code> .
	<code>y</code>	An instance of type <code>&lt;integer&gt;</code> .
Description	Returns the position of <code>caret</code> .	

See also      **`caret-sheet`**, page 191  
**`caret-size`**, page 191

## **`caret-sheet`** *Generic function*

Summary      Returns the sheet that owns the specified caret.  
Signature      **`cursor-sheet caret => sheet`**  
Arguments      **`caret`**      An instance of type `<caret>`.  
Values      **`sheet`**      An instance of type `<sheet>`.  
Description      Returns the sheet that owns *caret*.  
See also      **`caret-position`**, page 190  
**`caret-size`**, page 191

## **`caret-size`** *Generic function*

Summary      Returns the size of the specified caret.  
Signature      **`cursor-size caret => width height`**  
Arguments      **`caret`**      An instance of type `<caret>`.  
Values      **`width`**      An instance of type `<integer>`.  
                **`height`**      An instance of type `<integer>`.  
Description      Returns the size of *caret*.  
See also      **`caret-position`**, page 190  
**`caret-sheet`**, page 191

## **caret-visible?** *Generic function*

Summary	Returns true if the specified caret is visible.	
Signature	<code>cursor-visible? caret =&gt; visible?</code>	
Arguments	<code>caret</code>	An instance of type <code>&lt;caret&gt;</code> .
Values	<code>visible?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>caret</i> is visible.	
See also	<a href="#">&lt;cursor&gt;, page 211</a> <a href="#">caret-visible?-setter, page 192</a>	

## **caret-visible?-setter** *Generic function*

Summary	Specifies whether or not the specified caret is visible.	
Signature	<code>cursor-visible?-setter visible? caret =&gt; boolean</code>	
Arguments	<code>visible?</code>	An instance of type <code>&lt;boolean&gt;</code> .
	<code>caret</code>	An instance of type <code>&lt;caret&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Specifies whether or not <i>caret</i> is visible.	
See also	<a href="#">&lt;cursor&gt;, page 211</a> <a href="#">caret-visible?, page 192</a>	

## child-containing-position *Generic function*

Summary	Returns the topmost child of the specified sheet that occupies a specified position.	
Signature	<code>child-containing-position sheet x y =&gt; value</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>y</code>	An instance of type <code>&lt;real&gt;</code> .
Values	<code>value</code>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Returns the topmost enabled direct child of <code>sheet</code> whose region contains the position $(x, y)$ . The position is expressed in the coordinate system used by <code>sheet</code> .	
See also	<a href="#">children-overlapping-region, page 193</a> <a href="#">do-children-containing-position, page 223</a>	

## children-overlapping-region *Generic function*

Summary	Returns any children of the specified sheet whose regions overlap a specified region.	
Signature	<code>children-overlapping-region sheet region =&gt; sheets</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>region</code>	An instance of type <code>&lt;region&gt;</code> .
Values	<code>sheets</code>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;sheet&gt;)</code> .

Description      Returns the list of enabled direct children of *sheet* whose region overlaps *region*.

See also      [child-containing-position](#), page 193

[do-children-overlapping-region](#), page 223

## choose-color

*Generic function*

Summary      Displays the built-in color dialog for the target platform.

Signature      **choose-color #key frame owner title documentation exit-boxes name default => color**

Arguments      *frame*      An instance of type `<frame>`. Default value: `#f`.

*owner*      An instance of type `<sheet>`. Default value: `#f`.

*title*      An instance of type `<string>`.

*documentation*      An instance of type `<string>`.

*exit-boxes*      An instance of type `<object>`.

*name*      An instance of type `<object>`.

*default*      An instance of type `<object>`.

Values      *color*      An instance of type `<color>`.

Description      Displays the built-in color dialog for the target platform, which allows the user to choose a color from the standard palette for whatever environment the application is running in.

If the *frame* argument is specified, the top-level sheet of *frame* becomes the owner of the dialog.



**Figure 5.1** The standard Choose Color 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.

If you wish, you can specify a *title* for the dialog; this is displayed in the title bar of the frame containing the dialog.

#### Example

The following example illustrates how you can define a class of frame that contains a button that displays the Choose Color dialog, using the pre-built dialog classes for your target environment. The frame also contains an ellipse whose color is set to the color chosen from the dialog.

```

define frame <color-dialog-frame> (<simple-frame>)
  pane ellipse-pane (frame)
    make(<ellipse-pane>, foreground: $red);
  pane choose-color-button (frame)
    make(<menu-button>,
      label: "Choose Color...",
      documentation:
        "Example of standard 'choose color' dialog",
      activate-callback:
        method (button)
          let color = choose-color(owner: frame);
          color & change-ellipse-color(frame, color)
        end);
  end frame <color-dialog-frame>;

```

See also      [choose-directory](#), page 196

[choose-file](#), page 198

[notify-user](#), page 275

## choose-directory

*Generic function*

Summary      Displays the built-in directory dialog for the target platform.

Signature      **choose-directory #key frame owner title documentation exit-boxes name default => locator**

Arguments      **frame**      An instance of type <frame>. Default value: #f.

**owner**      An instance of type <sheet>. Default value: #f.

**title**      An instance of type <string>.

**documentation**      An instance of type <string>.

**exit-boxes**      An instance of type <object>.

**name**      An instance of type <object>.

**default**      An instance of type <object>.

Values	<i>locator</i>	An instance of type <code>type-union(&lt;string&gt;, &lt;locator&gt;)</code> .
Description	<p>Displays the built-in directory dialog for the target platform, which allows the user to choose a directory from any of the local or networked drives currently connected to the computer.</p> <p>If the <code>frame</code> argument is specified, the top-level sheet of <code>frame</code> becomes the owner of the dialog.</p> <p>Alternatively, you can specify the owner directly using the <code>owner</code> argument, which takes an instance of <code>&lt;sheet&gt;</code> as its value.</p> <p>By default, both <code>frame</code> and <code>owner</code> are <code>#f</code>, meaning the dialog has no owner. You should not specify both of these values.</p> <p>If you wish, you can specify a <code>title</code> for the dialog; this is displayed in the title bar of the frame containing the dialog.</p>	
Example	<p>The following example illustrates how you can define a class of frame that contains a button that displays the Choose Directory dialog, using the pre-built dialog classes for your target environment.</p>	

```

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
              ("Choose 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>;

```

See also      [choose-color](#), page 194  
[choose-file](#), page 198  
[notify-user](#), page 275

**choose-file***Generic function*

Summary	Displays the built-in file dialog for the target platform.	
Signature	<b>choose-file #key frame owner title documentation exit-boxes name default =&gt; locator</b>	
Arguments	<i>frame</i>	An instance of type < <b>frame</b> >. Default value: <b>#f</b> .
	<i>owner</i>	An instance of type < <b>sheet</b> >. Default value: <b>#f</b> .
	<i>title</i>	An instance of type < <b>string</b> >.
	<i>documentation</i>	An instance of type < <b>string</b> >.

<i>direction</i>	An instance of type <code>one-of(#"input", "#output")</code> . Default value: <code>#"input"</code> .
<i>filters</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;sequence&gt;)</code> .
<i>exit-boxes</i>	An instance of type <code>&lt;object&gt;</code> .
<i>name</i>	An instance of type <code>&lt;object&gt;</code> .
<i>default</i>	An instance of type <code>&lt;string&gt;</code> .
<b>Values</b>	<i>locator</i> An instance of type <code>&lt;string&gt;</code> .
<b>Description</b>	Displays the built-in file dialog for the target platform, which allows the user to choose a file from any of the local or networked drives currently connected to the computer. The function returns the name of the file chosen by the user.



**Figure 5.2** Typical appearance of a choose-file dialog

If the `frame` argument is 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.

If you wish, you can specify a *title* for the dialog; this is displayed in the title bar of the frame containing the dialog.

The *direction* argument is used to specify whether the file chosen is being opened (that is, information in the file is loaded into the application) or saved to (that is, information in the application is being saved to a file on disk).

The *filters* argument lets you specify the file filters that should be offered to the user in the dialog. These filters are typically available in a drop-down list box, and let the user display only certain types of file, such as text files. Each filter is described as a sequence of strings:

- The first string in the sequence is a description of the files that are displayed when this filter is chosen.
- Each subsequent string is a regular expression that describes which files to display in the dialog.

For example, to specify a filter that lets the user choose to display either text files, HTML files, or Dylan source files, the following sequence should be passed to the *filters* argument:

```
##[["Text files", "*.txt", "*text"],
  #[ "HTML files", "*.htm", "*html"],
  #[ "Dylan files", "*.dylan"]]
```

Here, text files are defined as any file with a filename suffix of `.txt` or `.text`, HTML files have filenames with a suffix of either `.htm` or `.html`, and Dylan files have filenames with a suffix of `.dylan`.

The *default* argument is used to specify a default filename to pass to the dialog. This is a convenient way to suggest a file in which some information may be saved, or a file to be loaded into an application.

**Example** The following example illustrates how you can define a class of frame that contains buttons to display both Open and Save As dialogs, using the pre-built dialog classes for your target environment.

```
define frame <open-save-dialog-frame> (<simple-frame>)
  pane open-file-button (frame)
    make(<menu-button>,
      label: "Open...",
      documentation:
        "Example of standard file 'Open' dialog",
      activate-callback:
        method (button)
          let file = choose-file(direction: #"input",
            owner: frame);
          if (file) frame-status-message(frame)
            := format-to-string
              ("Opened file %s", file);
        end
      end);
  pane save-file-button (frame)
    make(<menu-button>,
      label: "Save As...",
      documentation:
        "Example of standard file 'Save As' dialog",
      activate-callback:
        method (button)
          let file = choose-file(direction: #"output",
            owner: frame);
          if (file) frame-status-message(frame)
            := format-to-string
              ("Saved file as %s", file);
        end
      end);
end frame <open-save-dialog-frame>;
```

**See also**

- [choose-color](#), page 194
- [choose-directory](#), page 196
- [notify-user](#), page 275

	<b>choose-from-dialog</b>	<i>Generic function</i>
Summary	Prompt the user to choose from a collection of items, using a dialog box.	
Signature	<pre>choose-from-dialog <i>items</i> #key <i>frame owner title value default-item label-key value-key</i>       <i>selection-mode gadget-class gadget-options width height</i>       <i>foreground background text-style</i> =&gt; <i>value success?</i></pre>	
Arguments	<p><i>items</i> An instance of type-union(&lt;sequence&gt;, &lt;menu&gt;).</p> <p><i>frame</i> An instance of type &lt;frame&gt;. Default value: #f.</p> <p><i>owner</i> An instance of type &lt;sheet&gt;. Default value: #f.</p> <p><i>title</i> An instance of type &lt;string&gt;.</p> <p><i>default-item</i> An instance of type &lt;object&gt;.</p> <p><i>label-key</i> An instance of type &lt;function&gt;. Default value: identity.</p> <p><i>value-key</i> An instance of type &lt;function&gt;. Default value: identity.</p> <p><i>selection-mode</i> An instance of &lt;symbol&gt;. Default value: # "single".</p> <p><i>gadget-class</i> An instance of type &lt;gadget&gt;.</p> <p><i>gadget-options</i> An instance of type &lt;sequence&gt;.</p> <p><i>foreground</i> An instance of type &lt;ink&gt;.</p> <p><i>background</i> An instance of type &lt;ink&gt;.</p> <p><i>text-style</i> An instance of type &lt;text-style&gt;.</p>	

Values	<i>value</i>	An instance of type <object>.
	<i>success?</i>	An instance of type <boolean>.
Description	Prompt the user to choose from a collection of <i>items</i> , using a dialog box. This generic function is similar to <code>choose-from-menu</code> .	
	The function returns the values chosen by the user, and a boolean value: <code>#t</code> if a value was chosen, <code>#f</code> if nothing was chosen. Unlike <code>choose-from-menu</code> , the user can choose several values if desired, depending on the value of <i>selection-mode</i> , 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.

See also      [choose-from-menu, page 204](#)

## choose-from-menu

*Generic function*

Summary	Prompt the user to choose from a collection of items, using a pop-up menu.														
Signature	<pre>choose-from-menu <i>items</i>     #key <i>frame owner title value default-item label-key value-key</i>           <i>width height foreground background text-style multiple-sets?</i> =&gt; <i>value success?</i></pre>														
Arguments	<table> <tr> <td><i>items</i></td><td>An instance of <code>type-union(&lt;sequence&gt;, &lt;menu&gt;)</code>.</td></tr> <tr> <td><i>frame</i></td><td>An instance of type <code>&lt;frame&gt;</code>. Default value: <code>#f</code>.</td></tr> <tr> <td><i>owner</i></td><td>An instance of type <code>&lt;sheet&gt;</code>. Default value: <code>#f</code>.</td></tr> <tr> <td><i>title</i></td><td>An instance of type <code>&lt;string&gt;</code>. Default value: <code>#f</code>.</td></tr> <tr> <td><i>default-item</i></td><td>An instance of type <code>&lt;object&gt;</code>.</td></tr> <tr> <td><i>label-key</i></td><td>An instance of type <code>&lt;function&gt;</code>. Default value: <code>identity</code>.</td></tr> <tr> <td><i>value-key</i></td><td>An instance of type <code>&lt;function&gt;</code>. Default value: <code>identity</code>.</td></tr> </table>	<i>items</i>	An instance of <code>type-union(&lt;sequence&gt;, &lt;menu&gt;)</code> .	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> . Default value: <code>#f</code> .	<i>owner</i>	An instance of type <code>&lt;sheet&gt;</code> . Default value: <code>#f</code> .	<i>title</i>	An instance of type <code>&lt;string&gt;</code> . Default value: <code>#f</code> .	<i>default-item</i>	An instance of type <code>&lt;object&gt;</code> .	<i>label-key</i>	An instance of type <code>&lt;function&gt;</code> . Default value: <code>identity</code> .	<i>value-key</i>	An instance of type <code>&lt;function&gt;</code> . Default value: <code>identity</code> .
<i>items</i>	An instance of <code>type-union(&lt;sequence&gt;, &lt;menu&gt;)</code> .														
<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> . Default value: <code>#f</code> .														
<i>owner</i>	An instance of type <code>&lt;sheet&gt;</code> . Default value: <code>#f</code> .														
<i>title</i>	An instance of type <code>&lt;string&gt;</code> . Default value: <code>#f</code> .														
<i>default-item</i>	An instance of type <code>&lt;object&gt;</code> .														
<i>label-key</i>	An instance of type <code>&lt;function&gt;</code> . Default value: <code>identity</code> .														
<i>value-key</i>	An instance of type <code>&lt;function&gt;</code> . Default value: <code>identity</code> .														

	<i>foreground</i>	An instance of type < <code>ink</code> >.
	<i>background</i>	An instance of type < <code>ink</code> >.
	<i>text-style</i>	An instance of type < <code>text-style</code> >.
Values	<i>value</i>	An instance of type < <code>object</code> >.
	<i>success?</i>	An instance of type < <code>boolean</code> >.
Description	Prompt the user to choose from a collection of <i>items</i> , using a pop-up menu. This generic function is similar to <code>choose-from-dialog</code> .	The function returns the value chosen by the user, and a boolean value: <code>#t</code> if a value was chosen, <code>#f</code> if nothing was chosen.  At its most basic, <code>choose-from-menu</code> can be passed a simple sequence of items, as follows:
		<pre>choose-from-menu(#(1, 2, 3));</pre>
		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 <i>frame</i> keyword specifies a frame whose top level sheet becomes the owner of the menu. Alternatively, you can specify this top level sheet explicitly using <i>owner</i> . The <i>title</i> keyword lets you choose a title for the dialog. By default, each of these values is <code>#f</code> .
		In addition, <code>choose-from-menu</code> offers options similar to collection gadgets, that can act upon the items specified. The <i>default-item</i> keyword lets you specify an item that is returned by default if no value is chosen explicitly (thereby ensuring that <i>success?</i> will always be <code>#t</code> ). You can also specify a <i>value-key</i> or <i>label-key</i> for the items in the menu.
		Finally, you can configure the appearance of the menu itself. The <i>width</i> and <i>height</i> keywords let you set the size of the menu. The <i>foreground</i> and <i>background</i> keywords let you set the text color and the menu color respectively. The <i>text-style</i> keyword lets you specify a font to display the menu items.

See also      [choose-from-dialog](#), page 202

## **choose-text-style**

*Generic function*

Summary	Displays the built-in font dialog for the target platform, thereby letting the user choose a font.	
Signature	<code>choose-text-style #key frame owner title =&gt; font</code>	
Arguments	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> . Default value: <code>#f</code> .
	<i>owner</i>	An instance of type <code>&lt;sheet&gt;</code> . Default value: <code>#f</code> .
	<i>title</i>	An instance of type <code>&lt;string&gt;</code> . Default value: <code>#f</code> .
Values	<i>font</i>	An instance of <code>&lt;text-style&gt;</code> .
Description	<p>Displays the built-in font dialog for the target platform, thereby letting the user choose a font.</p> <p>The <i>frame</i> keyword specifies a frame whose top-level sheet becomes the owner of the menu. Alternatively, you can specify this top level sheet explicitly using <i>owner</i>. The <i>title</i> keyword lets you choose a title for the dialog. By default, each of these values is <code>#f</code>.</p> <p>If you wish, you can specify a <i>title</i> for the dialog; this is an instance of <code>&lt;string&gt;</code> and is displayed in the title bar of the frame containing the dialog. If you do not specify <i>title</i>, then DUIM uses the default title for that type of dialog on the target platform.</p>	

**clear-box** *Generic function*

**Summary** Clears a box-shaped area in the specified drawable.

**Signature**

```
clear-box  drawable left top right bottom => ()
clear-box* drawable region => ()
```

**Arguments**

<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
-----------------	--

The following arguments are specific to `clear-box`.

<i>left</i>	An instance of type <code>&lt;coordinate&gt;</code> .
<i>top</i>	An instance of type <code>&lt;coordinate&gt;</code> .
<i>right</i>	An instance of type <code>&lt;coordinate&gt;</code> .
<i>bottom</i>	An instance of type <code>&lt;coordinate&gt;</code> .

The following argument is specific to `clear-box*`.

<i>region</i>	An instance of type <code>&lt;region&gt;</code> .
---------------	---

**Values** None

**Description** Clears a box-shaped area in the specified drawable, removing anything that was drawn in that region.

The function `clear-box*` is identical to `clear-box`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

**clear-clipboard** *Generic function*

**Summary** Clears the contents of a clipboard.

**Signature**

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

Arguments	<i>clipboard</i>	An instance of <code>&lt;clipboard&gt;</code> .
Values	None.	
Description	Clears the contents of <i>clipboard</i> , which represents the locked clipboard.	

## `<clipboard>` *Open abstract class*

Summary	The class of clipboard objects.
Init-keywords	None.
Description	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 <code>&lt;clipboard&gt;</code> yourself, since this is handled automatically by the macro <code>with-clipboard</code> .
See also	<a href="#">add-clipboard-data</a> , page 184 <a href="#">add-clipboard-data-as</a> , page 185 <a href="#">clear-clipboard</a> , page 207 <a href="#">clipboard-data-available?</a> , page 209 <a href="#">clipboard-sheet</a> , page 209 <a href="#">clipboard-owner</a> , page 210 <a href="#">close-clipboard</a> , page 210 <a href="#">get-clipboard-data-as</a> , page 249 <a href="#">open-clipboard</a> , page 277 <a href="#">with-clipboard</a> , page 327

**clipboard-data-available?** *Generic function*

Summary	Returns false if there is any data of a particular type on a clipboard.	
Signature	<code>clipboard-data-available? <i>type clipboard</i> =&gt; <i>available?</i></code>	
Arguments	<i>type</i>	An instance of <code>type-union(&lt;symbol&gt;, &lt;type&gt;)</code> .
	<i>clipboard</i>	An instance of <code>&lt;clipboard&gt;</code> .
Values	<i>available?</i>	An instance of <code>&lt;boolean&gt;</code> .
Description	Returns <code>#f</code> if and only if there is any data of type <i>type</i> on the clipboard. The argument <i>type</i> is an instance of <code>type-union(&lt;symbol&gt;, &lt;type&gt;)</code> .	
See also	<a href="#">add-clipboard-data</a> , page 184 <a href="#">add-clipboard-data-as</a> , page 185 <a href="#">&lt;clipboard&gt;</a> , page 208 <a href="#">get-clipboard-data-as</a> , page 249	

**clipboard-sheet** *Generic function*

Summary	Returns the sheet with the clipboard lock.	
Signature	<code>clipboard-sheet <i>clipboard</i> =&gt; <i>sheet</i></code>	
	<i>clipboard</i>	An instance of <code>&lt;clipboard&gt;</code> .
Values	<i>sheet</i>	An instance of <code>&lt;sheet&gt;</code> .
Description	Returns the sheet with the clipboard lock.	
See also	<a href="#">&lt;clipboard&gt;</a> , page 208	

## clipboard-owner *Generic function*

Summary	Returns the sheet that owns the current clipboard data.	
Signature	<code>clipboard-owner clipboard =&gt; owner</code>	
Arguments	<code>clipboard</code>	An instance of <code>&lt;clipboard&gt;</code> .
Values	<code>owner</code>	An instance of <code>&lt;sheet&gt;</code> .
Description	Returns the sheet that owns the current clipboard data.	
See also	<code>&lt;clipboard&gt;</code> , page 208	

## close-clipboard *Function*

Summary	Closes the current clipboard lock for a sheet on a port.	
Signature	<code>close-clipboard port sheet =&gt; ()</code>	
Arguments	<code>port</code>	An instance of <code>&lt;port&gt;</code> .
	<code>sheet</code>	An instance of <code>&lt;sheet&gt;</code> .
Values	None.	
Description	<p>Closes the current clipboard lock for <i>sheet</i> on <i>port</i>. A clipboard lock needs to be closed safely after it the clipboard has been used, to free the clipboard for further use.</p> <p>You should not normally call <code>close-clipboard</code> yourself to close a clipboard lock. Use the macro <code>with-clipboard</code> to create and free the lock for you.</p>	
See also	<code>&lt;clipboard&gt;</code> , page 208 <code>with-clipboard</code> , page 327	

## \$control-key *Constant*

Summary	A constant that represents the CONTROL key on the keyboard.
Type	<code>&lt;integer&gt;</code>
Value	<code>ash(1, %modifier_base + 1);</code>
Description	A constant that represents the CONTROL key on the keyboard.
See also	<a href="#">\$alt-key</a> , page 185 <a href="#">\$hyper-key</a> , page 254 <a href="#">\$meta-key</a> , page 271 <a href="#">modifier-key-index</a> , page 273 <a href="#">modifier-key-index-name</a> , page 273 <a href="#">\$modifier-keys</a> , page 274 <a href="#">\$option-key</a> , page 278 <a href="#">\$shift-key</a> , page 317 <a href="#">\$super-key</a> , page 318

## <cursor> *Type*

Summary	The class of cursor objects.
Equivalent	<code>type-union(&lt;symbol&gt;, &lt;image&gt;)</code>
Init-keywords	None.

Description	The class of cursor objects. The cursor is the small image that is used to display the location of the mouse pointer at any time. A cursor can actually be any instance of <code>&lt;symbol&gt;</code> or any instance of <code>&lt;image&gt;</code> .
Operations	<code>pointer-cursor-setter</code> <code>set-caret-position</code> <code>sheet-pointer-cursor-setter</code>
See also	<code>&lt;caret&gt;</code> , page 189 <code>cursor?</code> , page 212

	<i>Generic function</i>
<b>cursor?</b>	
Summary	Returns true if the specified object is a cursor.
Signature	<code>cursor? object =&gt; cursor?</code>
Arguments	<code>object</code> An instance of type <code>&lt;object&gt;</code> .
Values	<code>cursor?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>object</i> is a cursor. In practice, you can create a cursor from any instance of <code>&lt;symbol&gt;</code> or <code>&lt;image&gt;</code> .
See also	<code>&lt;cursor&gt;</code> , page 211

	<i>Function</i>
<b>default-port</b>	
Summary	Returns the default port for the specified server.
Signature	<code>default-port #key server-path =&gt; port</code>
Arguments	<code>server-path</code> An instance of type <code>&lt;vector&gt;</code> . Default value: <code>#(#"local")</code> .

Values	<i>port</i>	An instance of type <code>false-or(&lt;port&gt;)</code> .
Description		Returns the default port for server specified by <i>server-path</i> .
See also		<code>default-port-setter</code> , page 213 <code>destroy-port</code> , page 213

## default-port-setter *Function*

Summary	Sets the default port.	
Signature	<code>default-port-setter port =&gt; port</code>	
Arguments	<i>port</i>	An instance of type <code>&lt;port&gt;</code> . Default value: <code>#f</code> .
Values	<i>port</i>	An instance of type <code>&lt;port&gt;</code> .
Description	Sets the default port.	
See also	<code>default-port</code> , page 212 <code>destroy-port</code> , page 213	

## destroy-port *Generic function*

Summary	Destroys the specified port.	
Signature	<code>destroy-port port =&gt; ()</code>	
Arguments	<i>port</i>	An instance of type <code>&lt;port&gt;</code> .
Values	None	
Description	Destroys <i>port</i> .	

See also **default-port**, page 212  
**default-port-setter**, page 213

## **destroy-sheet** *Generic function*

Summary	Destroys the specified sheet.	
Signature	<b>destroy-sheet</b> <i>sheet</i> => ()	
Arguments	<i>sheet</i>	An instance of type < <i>sheet</i> >.
Values	None	
Description	Destroys <i>sheet</i> .	

**<device-event>** *Open abstract class*

Summary	The class of device events.
Superclasses	< <b>sheet-event</b> >
Init-keywords	<b>sheet:</b> An instance of type < <b>sheet</b> >. <b>modifier-state:</b> An instance of type < <b>integer</b> >. Default value: 0.
Description	The class of device events. The <b>modifier-state:</b> init-keyword is used to record the state of the device at the time the event occurred.
Operations	The following operation is exported from the <b>DUIM-SHEET</b> module. <b>event-modifier-state</b>

**<display>***Open abstract class*

Summary	The class of displays.
Superclasses	<code>&lt;sheet&gt;</code>
Init-keywords	<p><b>orientation:</b> An instance of type <code>one-of(#"vertical", #"<b>horizontal</b>", #"<b>default</b>"). Default value: #"<b>default</b>".</code></p> <p><b>units:</b> An instance of type <code>one-of(#"<b>device</b>", #"<b>mm</b>", #"<b>pixels</b>"). Default value: #"<b>device</b>".</code></p>
Description	<p>The class of displays. An instance of <code>&lt;display&gt;</code> is an object that represents a single display (or screen) on some display server. Any sheet can be attached to an instance of <code>&lt;display&gt;</code>, and a display, and all the sheets attached to it, are associated with a <code>&lt;port&gt;</code> that is a connection to a display server.</p> <p>The <code>orientation:</code> init-keyword is used to specify the orientation of a display.</p> <p>The <code>units:</code> init-keyword is used to specify the units in which height and width measurements are made with respect to the display. The default is whatever units are standard for the display device (usually pixels).</p>
Operations	<p>The following operations are exported from the <code>DUIM-Sheets</code> module.</p> <pre>display display? display-depth display-height display- mm-height display-mm-width display-orientation display-pixel-height display-pixels-per-point dis- play-pixel-width display-units display-width</pre>
See also	<p><code>display</code>, page 216</p> <p><code>display?</code>, page 216</p>

`display-depth`, page 217  
`display-height`, page 217  
`display-orientation`, page 219  
`display-units`, page 221  
`display-width`, page 222  
`<port>`, page 287  
`<sheet>`, page 299

**display** *Generic function*

Summary	Returns the display for the specified object.	
Signature	<code>display object =&gt; display</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>display</code>	An instance of type <code>false-or(&lt;display&gt;)</code> .
Description	Returns the display used to display <i>object</i> .	
See also	<code>&lt;display&gt;</code> , page 215 <code>frame-manager</code> , page 244 <code>port</code> , page 288	

**display?** *Generic function*

Summary	Returns true if the specified object is a display.	
Signature	<code>display? object =&gt; display?</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .

Values	<i>display?</i>	An instance of type <boolean>.
Description	Returns true if <i>object</i> is a display.	
See also	< <i>display</i> >, page 215	

## display-depth

*Generic function*

Summary	Returns the color depth of the specified display.	
Signature	<b>display-depth</b> <i>display</i> => <i>depth</i>	
Arguments	<i>display</i>	An instance of type < <i>display</i> >.
Values	<i>depth</i>	An instance of type <integer>.
Description	Returns the color depth of <i>display</i> . By default, the color depth of any display is assumed to be 8.	
See also	<a href="#">display-height</a> , page 217 <a href="#">display-orientation</a> , page 219 <a href="#">display-width</a> , page 222	

## display-height

*Generic function*

Summary	Returns the height of the specified display.	
Signature	<b>display-height</b> <i>display</i> #key <i>units</i> => <i>height</i>	
Arguments	<i>display</i>	An instance of type < <i>display</i> >.
	<i>units</i>	An instance of one-of(#"device", #"mm", #"pixels"). Default value: #"device".

Values	<i>height</i>	An instance of type <number>.
Description	Returns the height of <i>display</i> , in device-independent units. If <i>units</i> is specified, then the value returned is converted into the appropriate type of units.	
See also	<a href="#">display-depth</a> , page 217 <a href="#">display-mm-height</a> , page 218 <a href="#">display-orientation</a> , page 219 <a href="#">display-pixel-height</a> , page 220 <a href="#">display-units</a> , page 221 <a href="#">display-width</a> , page 222	

## display-mm-height *Generic function*

Summary	Returns the height of the specified display in millimeters.	
Signature	<code>display-mm-height <i>display</i> =&gt; <i>height</i></code>	
Arguments	<i>display</i>	An instance of type <display>.
Values	<i>height</i>	An instance of type <number>.
Description	Returns the height of <i>display</i> in millimeters. This is equivalent to calling <code>display-height</code> with the <i>units</i> argument set to <code>#"mm"</code> .	
See also	<a href="#">display-height</a> , page 217 <a href="#">display-mm-width</a> , page 219 <a href="#">display-pixel-height</a> , page 220 <a href="#">display-units</a> , page 221	

## display-mm-width *Generic function*

Summary	Returns the width of the specified display in millimeters.
Signature	<code>display-mm-width <i>display</i> =&gt; <i>width</i></code>
Arguments	<i>display</i> An instance of type <code>&lt;display&gt;</code> .
Values	<i>width</i> An instance of type <code>&lt;number&gt;</code> .
Description	Returns the width of <i>display</i> in millimeters. This is equivalent to calling <code>display-width</code> with the <i>units</i> argument set to <code>#"mm"</code> .
See also	<a href="#">display-mm-height</a> , page 218 <a href="#">display-pixel-width</a> , page 221 <a href="#">display-units</a> , page 221 <a href="#">display-width</a> , page 222

## display-orientation *Generic function*

Summary	Returns the orientation of the specified display.
Signature	<code>display-orientation <i>display</i> =&gt; <i>orientation</i></code>
Arguments	<i>display</i> An instance of type <code>&lt;display&gt;</code> .
Values	<i>orientation</i> An instance of type <code>one-of(#"vertical", "#horizontal", #"<b>default</b>")</code> .
Description	Returns the orientation of <i>display</i> . Unless specified otherwise, the orientation of any display is <code>#"<b>default</b>"</code> .
See also	<a href="#">display-depth</a> , page 217

`display-height`, page 217

`display-width`, page 222

## display-pixel-height

*Generic function*

Summary      Returns the height of the specified display in pixels.

Signature      `display-pixel-height display => height`

Arguments      `display`      An instance of type `<display>`.

Values      `height`      An instance of type `<integer>`.

Description      Returns the height of `display` in pixels. This is equivalent to calling `display-height` with the `units` argument set to `#"pixels"`.

See also      `display-height`, page 217

`display-mm-height`, page 218

`display-pixel-width`, page 221

`display-units`, page 221

## display-pixels-per-point

*Generic function*

Summary      Returns the number of pixels per point for the specified display.

Signature      `display-pixels-per-point display => number`

Arguments      `display`      An instance of type `<display>`.

Values      `number`      An instance of type `<number>`.

Description     Returns the number of pixels per point for *display*.

See also     `display-pixel-height`, page 220

`display-pixel-width`, page 221

`display-units`, page 221

## **display-pixel-width**

*Generic function*

Summary     Returns the width of the specified display in pixels.

Signature     `display-pixel-width display => width`

Arguments     `display`     An instance of type `<display>`.

Values     `width`     An instance of type `<integer>`.

Description     Returns the height of *display* in pixels. This is equivalent to calling `display-width` with the *units* argument set to `#"pixels"`.

See also     `display-mm-width`, page 219

`display-pixel-height`, page 220

`display-units`, page 221

`display-width`, page 222

## **display-units**

*Generic function*

Summary     Returns the default units for the specified display.

Signature     `display-units display => value`

Arguments     `display`     An instance of type `<display>`.

Values	<i>value</i>	An instance of type <code>one-of(#"device", #"<b>pixels</b>", #"<b>mm</b>").</code>
Description		Returns the default units for <i>display</i> . These are the units in which height and width measurements are made, both for the display, and for any children of the display. Unless otherwise specified, the value returned is <code>#"<b>default</b>"</code> , so as to maintain a device-independent measurement as far as possible.
See also		<code>display-height</code> , page 217 <code>display-width</code> , page 222

## **display-width** *Generic function*

Summary	Returns the width of the specified display.	
Signature	<code>display-width <i>display</i> #key <i>units</i> =&gt; <i>width</i></code>	
Arguments	<i>display</i>	An instance of type <code>&lt;display&gt;</code> .
	<i>units</i>	An instance of <code>one-of(#"device", #"<b>mm</b>", #"<b>pixels</b>")</code> . Default value: <code>#"<b>device</b>"</code> .
Values	<i>width</i>	An instance of type <code>&lt;number&gt;</code> .
Description	Returns the width of <i>display</i> , in device-independent units. If <i>units</i> is specified, then the value returned is converted into the appropriate type of units.	
See also	<code>display-depth</code> , page 217 <code>display-height</code> , page 217 <code>display-mm-width</code> , page 219 <code>display-orientation</code> , page 219	

`display-pixel-width`, page 221

`display-units`, page 221

## do-children-containing-position *Generic function*

**Summary** Invokes a function on any children that occupy a specified position in the specified sheet.

**Signature** `do-children-containing-position function sheet x y => ()`

**Arguments** *function* An instance of type `<function>`.

*sheet* An instance of type `<sheet>`.

*x* An instance of type `<real>`.

*y* An instance of type `<real>`.

**Values** None

**Description** Invokes *function* on any children that occupy position (*x*, *y*) in *sheet*. This is used by `child-containing-position` to ascertain which children occupy the position. The function `child-containing-position` then decides which of the children returned is the topmost direct enabled child.

**See also** `child-containing-position`, page 193

## do-children-overlapping-region *Generic function*

**Summary** Invokes a function on any children of the specified sheet whose regions overlap a specified region.

**Signature** `do-children-overlapping-region function sheet region => ()`

**Arguments** *function* An instance of type `<function>`.

<i>sheet</i>	An instance of type < <code>sheet</code> >.
<i>region</i>	An instance of type < <code>region</code> >.
Values	None
Description	Invokes <i>function</i> on any children of <i>sheet</i> whose regions overlap <i>region</i> . This is used by <code>children-overlapping-region</code> to ascertain which children overlap <i>region</i> .
See also	<code>children-overlapping-region</code> , page 193 <code>do-children-containing-position</code> , page 223

## do-displays *Inline function*

Summary	Runs a function on all the displays attached to a given port.	
Signature	<code>do-displays function port =&gt; ()</code>	
Arguments	<i>function</i>	An instance of type < <code>function</code> >.
	<i>port</i>	An instance of type < <code>port</code> >.
Values	None	
Description	Runs a function on all the displays attached to a given port. By default, the current port is used, unless <i>port</i> is specified.	

## do-frames *Generic function*

Summary	Runs a function on all the frames managed by a given frame manager.	
Signature	<code>do-frames function #key port frame-manager =&gt; ()</code>	
Arguments	<i>function</i>	An instance of type < <code>function</code> >.

<i>port</i>	An instance of type < <b>port</b> >.
<i>frame-manager</i>	An instance of type < <b>frame-manager</b> >.
Values	None
Description	Runs a function on all the frames managed by a given frame manager. By default, the current frame manager on the current port is used, unless <i>port</i> or <i>frame-manager</i> are specified.

## do-ports *Function*

Summary	Runs a function on all the current ports.	
Signature	<b>do-ports</b> <i>function</i> => ()	
Arguments	<i>function</i>	An instance of type < <b>function</b> >.
Values	None	
Description	Runs a function on all the current ports.	

## do-sheet-children *Generic function*

Summary	Runs a function on all the immediate children of the specified sheet.	
Signature	<b>do-sheet-children</b> <i>function sheet</i> => ()	
Arguments	<i>function</i>	An instance of type < <b>function</b> >.
	<i>sheet</i>	An instance of type < <b>sheet</b> >.
Values	None	

Description      Runs *function* on all the immediate children of *sheet*. This function calls `sheet-children` to find the children of *sheet*.

See also      `sheet-children`, page 305

## **do-sheet-tree** *Generic function*

Summary      Runs a function on all the children in the hierarchy of the specified sheet.

Signature      `do-sheet-tree function sheet => ()`

Arguments      *function*      An instance of type `<function>`.  
*sheet*      An instance of type `<sheet>`.

Values      None

Description      Runs a function on all the children in the hierarchy of the specified sheet. The function is run on *sheet*, then on the children of *sheet*, then on the children of the children of *sheet*, and so on.

## **<double-click-event>** *Sealed instantiable class*

Summary      The class of double-click events on the pointer device.

Superclasses      `<button-press-event>`

Init-keywords      None.

Description      The class of double-click events on the pointer device. An instance of this class is generated when a button press is detected within a certain (small) amount of time after a previ-

ous button press. If a double click event is generated, the clock is reset, so that the next press generated is an instance of `<button-press-event>`.

Operations	None.
See also	<code>&lt;button-press-event&gt;</code> , page 188

## do-with-drawing-options *Generic function*

Summary	Runs some code on a drawable in a given drawing context.	
Signature	<code>do-with-drawing-options</code> <i>drawable</i> <i>function</i> #key <i>brush</i> <i>pen</i> <i>text-style</i> <i>clipping-region</i> <i>transform</i> => #rest <i>values</i>	
Arguments	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	<i>function</i>	An instance of type <code>&lt;function&gt;</code> .
	<i>brush</i>	An instance of type <code>&lt;brush&gt;</code> .
	<i>pen</i>	An instance of type <code>&lt;pen&gt;</code> .
	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
	<i>clipping-region</i>	An instance of type <code>&lt;region&gt;</code> .
	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
Values	<i>values</i>	An instance of type <code>&lt;object&gt;</code> .
Description	<p>Runs some code on a drawable in a given drawing context. This function is called by the macro <code>with-drawing-options</code>, and you should define new methods on it for new classes of drawable.</p> <p>The <i>function</i> passed to <code>do-with-drawing-options</code> is the result of encapsulating the body passed to <code>with-drawing-options</code> as a stand-alone method.</p>	

The values returned are the values that are returned from `with-drawing-options`.

The various keywords specify a drawing context in which function is run.

See also      `with-drawing-options`, page 329

## **do-with-pointer-grabbed** *Generic function*

**Summary**      Runs some specified code, forwarding all pointer events to a sheet.

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

**Arguments**

<code>port</code>	An instance of type <code>&lt;port&gt;</code> .
<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
<code>continuation</code>	An instance of type <code>&lt;function&gt;</code> .

**Values**      `values`      An instance of type `<object>`.

**Description**      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>`.

This function 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`.

See also      `with-pointer-grabbed`, page 333

**do-with-sheet-medium***Generic function*

Summary	Runs a continuation function on a sheet.	
Signature	<code>do-with-sheet-medium sheet continuation =&gt; #rest values</code>	
Arguments	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
	<i>continuation</i>	An instance of type <code>&lt;function&gt;</code> .
Values	<i>values</i>	An instance of type <code>&lt;object&gt;</code> .
Description	Runs a continuation function on a sheet.	
See also	<a href="#">with-sheet-medium</a> , page 335	

**do-with-text-style***Generic function*

Summary	Runs some code on a drawable in the context of a given text style.	
Signature	<code>do-with-text-style drawable function text-style =&gt; ()</code>	
Arguments	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	<i>function</i>	An instance of type <code>&lt;function&gt;</code> .
	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Values	None	
Description	Runs some code on a drawable in the context of a given text style.	
See also	<a href="#">with-text-style</a> , page 336	

	<i>Generic function</i>						
Summary	Returns the result of running a function in a transform defined on a specified medium.						
Signature	<code>do-with-transform</code> <i>drawable function transform =&gt; #rest values</i>						
Arguments	<table> <tr> <td><i>drawable</i></td><td>An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</td></tr> <tr> <td><i>function</i></td><td>An instance of type <code>&lt;function&gt;</code>.</td></tr> <tr> <td><i>transform</i></td><td>An instance of type <code>&lt;transform&gt;</code>.</td></tr> </table>	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .	<i>function</i>	An instance of type <code>&lt;function&gt;</code> .	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .						
<i>function</i>	An instance of type <code>&lt;function&gt;</code> .						
<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .						
Values	<i>values</i> An instance of type <code>&lt;object&gt;</code> .						
Description	Returns the result of running a function in a transform defined on a specified medium. Methods on this function are called by <code>with-transform</code> , which in turn is used by the similar macros <code>with-rotation</code> , <code>with-scaling</code> , and <code>with-translation</code> .						
See also	<code>with-transform</code> , page 337						

	<i>Open abstract class</i>
Summary	The base class of all DUIM events.
Superclasses	<code>&lt;object&gt;</code>
Init-keywords	<code>timestamp:</code> An instance of type <code>&lt;integer&gt;</code> . Default value: <code>next-event-timestamp()</code> .
Description	<p>The base class of all DUIM events.</p> <p>The <code>timestamp:</code> init-keyword is used to give a unique identifier for the event.</p>

**Operations** The following operations are exported from the `DUIM-Sheets` module.

```
event? event-matches-gesture? handle-event queue-event
```

**See also** `<frame-event>`, page 243  
`<sheet-event>`, page 307

## event?

*Generic function*

**Summary** Returns true if the specified object is an event.

**Signature** `event? object => event?`

**Arguments** `object` An instance of type `<object>`.

**Values** `event?` An instance of type `<boolean>`.

**Description** Returns true if `object` is an instance of `<event>` or one of its subclasses.

**See also** `<event>`, page 230

## event-button

*Generic function*

**Summary** Returns an integer corresponding to the mouse button that was pressed or released.

**Signature** `event-button event => integer`

**Arguments** `event` An instance of type `<event>`.

**Values** `integer` An instance of type `<integer>`.

Description	Returns an integer corresponding to the mouse button that was pressed or released, which will be one of <code>\$left-button</code> , <code>\$middle-button</code> , or <code>\$right-button</code> .  <b>Note:</b> The function <code>event-button</code> records the button state at the time that the event occurred, and hence can be different from <code>pointer-button-state</code> .
See also	<a href="#"><code>\$left-button</code>, page 257</a> <a href="#"><code>\$middle-button</code>, page 272</a> <a href="#"><code>&lt;pointer-button-event&gt;</code>, page 280</a> <a href="#"><code>pointer-button-state</code>, page 282</a> <a href="#"><code>\$right-button</code>, page 295</a>

	<i>Generic function</i>
Summary	Returns the character that was pressed on the keyboard.
Signature	<code>event-character event =&gt; value</code>
Arguments	<code>event</code> An instance of type <code>&lt;event&gt;</code> .
Values	<code>value</code> An instance of type <code>false-or(&lt;character&gt;)</code> .
Description	Returns the character associated with the keyboard event, if there is any.
See also	<a href="#"><code>event-key-name</code>, page 233</a> <a href="#"><code>&lt;keyboard-event&gt;</code>, page 254</a>

**event-key-name** *Generic function*

Summary	Returns the name of the key that was pressed or released on the keyboard.	
Signature	<code>event-key-name event =&gt; name</code>	
Arguments	<code>event</code>	An instance of type <code>&lt;event&gt;</code> .
Values	<code>name</code>	An instance of type <code>&lt;symbol&gt;</code> .
Description	Returns the name of the key that was pressed or released in a keyboard event. This will be a symbol whose value is specific to the current port.	
See also	<a href="#">event-character</a> , page 232 <a href="#">&lt;keyboard-event&gt;</a> , page 254	

**event-matches-gesture?** *Generic function*

Summary	Returns true if an event matches a defined gesture.	
Signature	<code>event-matches-gesture? event gesture-name =&gt; matches?</code>	
Arguments	<code>event</code>	An instance of type <code>&lt;event&gt;</code> .
	<code>gesture-name</code>	An instance of type <code>type-union(&lt;gesture&gt;, &lt;character&gt;)</code> .
Values	<code>matches?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if an event matches a defined gesture.	

<b>event-modifier-state</b>		<i>Generic function</i>
Summary	Returns an integer value that encodes the state of all the modifier keys on the keyboard.	
Signature	<code>event-modifier-state event =&gt; integer</code>	
Arguments	<code>event</code>	An instance of type <code>&lt;event&gt;</code> .
Values	<code>integer</code>	An instance of type <code>&lt;integer&gt;</code> .
Description	Returns an integer value that encodes the state of all the modifier keys on the keyboard. This is a mask consisting of the logior of <code>\$shift-key</code> , <code>\$control-key</code> , <code>\$meta-key</code> , <code>\$super-key</code> , and <code>\$hyper-key</code> .	
See also	<a href="#">event-sheet</a> , page 235 <a href="#">gesture-modifier-state</a> , page 248 <a href="#">make-modifier-state</a> , page 258 <a href="#">port-modifier-state</a> , page 289	

<b>event-pointer</b>		<i>Generic function</i>
Summary	Returns the pointer object to which the specified event refers.	
Signature	<code>event-pointer event =&gt; pointer</code>	
Arguments	<code>event</code>	An instance of type <code>&lt;event&gt;</code> .
Values	<code>pointer</code>	An instance of type <code>&lt;pointer&gt;</code> .
Description	Returns the pointer object to which <code>event</code> refers.	
See also	<a href="#">&lt;pointer&gt;</a> , page 278	

**event-x**, page 236

**event-y**, page 236

## event-region

### *Generic function*

Summary	Returns the region in the sheet that is affected by the specified event.	
Signature	<b>event-region</b> <i>event</i> => <i>region</i>	
Arguments	<i>event</i>	An instance of type < <i>event</i> >.
Values	<i>region</i>	An instance of type < <i>region</i> >.
Description	Returns the region of the sheet that is affected by <i>event</i> .	
See also	<a href="#">event-x</a> , page 236 <a href="#">event-y</a> , page 236 <a href="#">&lt;window-event&gt;</a> , page 325	

## event-sheet

### *Generic function*

Summary	Returns the sheet associated with the specified event.	
Signature	<b>event-sheet</b> <i>event</i> => <i>sheet</i>	
Arguments	<i>event</i>	An instance of type < <i>event</i> >.
Values	<i>sheet</i>	An instance of type < <i>sheet</i> >.
Description	Returns the sheet associated with <i>event</i> .	
See also	<a href="#">event-modifier-state</a> , page 234	

	<i>Generic function</i>
<b>event-x</b>	
Summary	Returns the x position of the pointer at the time the event occurred.
Signature	<code>event-x event =&gt; x</code>
Arguments	<code>event</code> An instance of type <code>&lt;event&gt;</code> .
Values	<code>x</code> An instance of type <code>&lt;integer&gt;</code> .
Description	Returns the x position of the pointer at the time the event occurred, in the coordinate system of the sheet that received the event.
See also	<a href="#">event-pointer</a> , page 234 <a href="#">event-region</a> , page 235 <a href="#">event-y</a> , page 236

	<i>Generic function</i>
<b>event-y</b>	
Summary	Returns the y position of the pointer at the time the event occurred.
Signature	<code>event-y event =&gt; y</code>
Arguments	<code>event</code> An instance of type <code>&lt;event&gt;</code> .
Values	<code>y</code> An instance of type <code>&lt;integer&gt;</code> .
Description	Returns the y position of the pointer at the time the event occurred, in the coordinate system of the sheet that received the event.
See also	<a href="#">event-pointer</a> , page 234

**event-region**, page 235

**event-x**, page 236

## find-display *Function*

Summary	Returns a suitable display for the specified port and server-path criteria.	
Signature	<b>find-display</b> #key <i>server-path</i> <i>port</i> <i>orientation</i> <i>units</i> => <i>display</i>	
Arguments	<i>server-path</i>	An instance of type <symbol>. Default value: #(#"local").
	<i>port</i>	An instance of type <port>.
	<i>orientation</i>	An instance of type one-of(#"default"). Default value: #"default".
	<i>units</i>	An instance of type one-of(#"device", #"pixels", #"mm"). Default value: #"device".
Values	<i>display</i>	An instance of type <display>.
Description	Returns a suitable display for the specified port and server-path criteria.  The <i>orientation</i> and <i>units</i> arguments can be used to specify the orientation and display units that the returned <i>display</i> needs to use.	
See also	<b>find-port</b> , page 238	

## find-frame-manager *Function*

Summary	Returns a suitable frame manager for the specified criteria.
---------	--

Signature	<b>find-frame-manager #rest options #key port server-path class palette =&gt; framem</b>	
Arguments	<i>options</i>	An instance of type <object>.
	<i>port</i>	An instance of type <port>.
	<i>server-path</i>	An instance of type <object>.
	<i>class</i>	An instance of type <type>.
	<i>palette</i>	An instance of type <palette>.
Values	<i>framem</i>	An instance of type <frame-manager>.
Description	<p>Returns a suitable frame manager for the specified criteria. If necessary, you can specify a <i>port</i>, <i>server-path</i>, <i>class</i>, or <i>palette</i>. If any of these are not specified, then the default value is used in each case. The <i>class</i> argument specifies the class of frame manager that should be returned.</p>	

		<i>Function</i>
	<b>find-port</b>	<i>Function</i>
Summary	Returns a suitable port for the specified server-path.	
Signature	<b>find-port #rest initargs #key server-path =&gt; port</b>	
Arguments	<i>initargs</i>	An instance of type <object>.
	<i>server-path</i>	An instance of type <object>. Default value: <b>*default-server-path*</b> .
Values	<i>port</i>	An instance of type <port>.
Description	Returns a suitable port for the specified server-path.	
See also	<a href="#">find-display</a> , page 237	

## **fixed-width-font?**

*Generic function*

Summary	Returns true if the specified text style uses a fixed-width font.	
Signature	<code>fixed-width-font? <i>text-style</i> <i>port</i> #key <i>character-set</i> =&gt; <i>fixed?</i></code>	
Arguments	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
	<i>port</i>	An instance of type <code>&lt;port&gt;</code> .
	<i>character-set</i>	An instance of type <code>&lt;object&gt;</code> . Default value: <code>\$standard-character-set</code> .
Values	<i>fixed?</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>text-style</i> uses a fixed-width font.	

## **font-ascent**

*Generic function*

Summary	Returns the ascent of the font in the specified text style.	
Signature	<code>font-ascent <i>text-style</i> <i>port</i> #key <i>character-set</i> =&gt; <i>ascent</i></code>	
Arguments	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
	<i>port</i>	An instance of type <code>&lt;port&gt;</code> .
	<i>character-set</i>	An instance of type <code>&lt;object&gt;</code> . Default value: <code>\$standard-character-set</code> .
Values	<i>ascent</i>	An instance of type <code>&lt;real&gt;</code> .
Description	Returns the ascent of the font in the <i>text-style</i> on <i>port</i> .	
See also	<a href="#">font-descent</a> , page 240 <a href="#">font-height</a> , page 240 <a href="#">font-metrics</a> , page 241	

**font-width**, page 242

## **font-descent** *Generic function*

Summary      Returns the descent of the font in the specified text style.

Signature     **font-descent** *text-style* *port* #key *character-set* => *descent*

Arguments    *text-style*      An instance of type <**text-style**>.

*port*         An instance of type <**port**>.

*character-set*   An instance of type <**object**>.

Values        *descent*      An instance of type <**real**>.

Description    Returns the descent of the font in the *text-style* on *port*.

See also     **font-ascent**, page 239

**font-height**, page 240

**font-metrics**, page 241

**font-width**, page 242

## **font-height** *Generic function*

Summary      Returns the height of the font in the specified text style.

Signature     **font-height** *text-style* *port* #key *character-set* => *height*

Arguments    *text-style*      An instance of type <**text-style**>.

*port*         An instance of type <**port**>.

*character-set*   An instance of type <**object**>.

Values        *height*      An instance of type <**real**>.

Description	Returns the height of the font in the <i>text-style</i> on <i>port</i> .
See also	<a href="#">font-ascent</a> , page 239 <a href="#">font-descent</a> , page 240 <a href="#">font-metrics</a> , page 241 <a href="#">font-width</a> , page 242

## font-metrics

## *Generic function*

Summary	Returns the metrics of the font in the specified text style.
Signature	<b>font-metrics</b> <i>text-style</i> <i>port</i> #key <i>character-set</i> => <i>font width height ascent descent</i>
Arguments	<i>text-style</i> An instance of type <text-style>. <i>port</i> An instance of type <port>. <i>character-set</i> An instance of type <object>.
Values	<i>font</i> An instance of type <object>. <i>width</i> An instance of type <real>. <i>height</i> An instance of type <real>. <i>ascent</i> An instance of type <real>. <i>descent</i> An instance of type <real>.
Description	Returns the metrics of the font in the <i>text-style</i> on <i>port</i> .
See also	<b>font-ascent</b> , page 239 <b>font-descent</b> , page 240 <b>font-height</b> , page 240 <b>font-width</b> , page 242

		<i>Generic function</i>
<b>font-width</b>		
Summary	Returns the width of the font in the specified text style.	
Signature	<code>font-width <i>text-style</i> <i>port</i> #key <i>character-set</i> =&gt; <i>width</i></code>	
Arguments	<p><i>text-style</i>      An instance of type <code>&lt;text-style&gt;</code>.</p> <p><i>port</i>            An instance of type <code>&lt;port&gt;</code>.</p> <p><i>character-set</i>    An instance of type <code>&lt;object&gt;</code>.</p>	
Values	<i>width</i> An instance of type <code>&lt;real&gt;</code> .	
Description	Returns the width of the font in the <i>text-style</i> on <i>port</i> .	
See also	<a href="#">font-ascent</a> , page 239 <a href="#">font-descent</a> , page 240 <a href="#">font-height</a> , page 240 <a href="#">font-metrics</a> , page 241	
<b>force-display</b>		<i>Generic function</i>
Summary	Forces the specified drawable object to be displayed.	
Signature	<code>force-display <i>drawable</i> =&gt; ()</code>	
Arguments	<i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .	
Values	None	
Description	Forces <i>drawable</i> to be displayed.	

**<frame-event>***Open abstract class*

Summary	The class of events that occur in frames.
Superclasses	<code>&lt;event&gt;</code>
Init-keywords	<code>frame:</code> An instance of type <code>&lt;frame&gt;</code> . Required.
Description	The class of events that occur in frames. The <code>frame:</code> init-keyword specified the frame in which the event occurs.
Operations	None.
See also	<a href="#"><code>&lt;frame-created-event&gt;</code>, page 702</a> <a href="#"><code>&lt;frame-destroyed-event&gt;</code>, page 702</a> <a href="#"><code>&lt;frame-exited-event&gt;</code>, page 704</a> <a href="#"><code>&lt;frame-exit-event&gt;</code>, page 705</a> <a href="#"><code>&lt;frame-mapped-event&gt;</code>, page 711</a> <a href="#"><code>&lt;frame-unmapped-event&gt;</code>, page 724</a>

**<frame-manager>***Open abstract class*

Summary	The class of frame managers.
Superclasses	<code>&lt;object&gt;</code>
Init-keywords	None.
Description	<p>The class of frame managers.</p> <p>Frame managers control the realization of the look and feel of a frame. The frame manager interprets the specification of the application frame in the context of the available window system facilities, taking into account preferences expressed by the user.</p>

In addition, the frame manager takes care of attaching the pane hierarchy of an application frame to an appropriate place in a window hierarchy.

Thus, the frame manager decides the following:

- What concrete gadget to create for an abstract gadget.
- How to layout the various parts of a frame, such as its menu, tool, and status bars.
- How to lay out dialogs and their exit buttons.
- How much spacing to use in various conventional layouts.

In addition, a frame manager maps dialog functions such as `choose-file` to their appropriate native dialogs.

Operations      The following operations are exported from the **DUIM-Sheets** module.

```
display frame-manager? frame-manager-frames frame-
manager-palette frame-manager-palette-setter port
```

The following operations are exported from the **DUIM-Frames** module.

```
clear-progress-note display-progress-note make-menus-
from-command-table
```

The following operation is exported from the **DUIM-DCS** module.

```
find-color
```

See also      **frame-manager**, page 244

**frame-manager?**, page 245

## **frame-manager**

*Generic function*

Summary      Returns the frame manager for the specified object.

Signature	<code>frame-manager object =&gt; value</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>value</code>	An instance of type <code>false-or (&lt;frame-manager&gt;)</code> .
Description	Returns the frame manager used to control the look and feel of the display of <i>object</i> .	
See also	<a href="#">display</a> , page 216 <a href="#">&lt;frame-manager&gt;</a> , page 243 <a href="#">frame-manager?</a> , page 245 <a href="#">port</a> , page 288	

## frame-manager? *Generic function*

Summary	Returns true if the specified object is a frame manager.	
Signature	<code>frame-manager? object =&gt; framem?</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>framem?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>object</i> is a frame manager.	
See also	<a href="#">&lt;frame-manager&gt;</a> , page 243 <a href="#">frame-manager</a> , page 244	

## frame-manager-frames *Generic function*

Summary	Returns the frames managed by the specified frame manager.
---------	--

Signature	<code>frame-manager-frames framem =&gt; frames</code>	
Arguments	<code>framem</code>	An instance of type <code>&lt;frame-manager&gt;</code> .
Values	<code>frames</code>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;frame&gt;)</code> .
Description	Returns the frames managed by <i>framem</i> .	

## **frame-manager-palette** *Generic function*

Summary	Returns the palette used by the specified frame manager.	
Signature	<code>frame-manager-palette framem =&gt; palette</code>	
Arguments	<code>framem</code>	An instance of type <code>&lt;frame-manager&gt;</code> .
Values	<code>palette</code>	An instance of type <code>&lt;palette&gt;</code> .
Description	Returns the palette used by <i>framem</i> .	
See also	<a href="#">frame-manager-palette-setter, page 246</a>	

## **frame-manager-palette-setter** *Generic function*

Summary	Sets the palette used by the specified frame manager.	
Signature	<code>frame-manager-palette-setter palette framem =&gt; palette</code>	
Arguments	<code>palette</code>	An instance of type <code>&lt;palette&gt;</code> .
	<code>framem</code>	An instance of type <code>&lt;frame-manager&gt;</code> .
Values	<code>palette</code>	An instance of type <code>&lt;palette&gt;</code> .

Description Sets the palette used by *framem*.

See also [frame-manager-palette](#), page 246

## **<gesture>** *Abstract instantiable class*

Summary The base class of all gestures.

Superclasses **<object>**

Init-keywords **keysym:** An instance of type **<symbol>**. Required.  
**button:** An instance of type **<integer>**. Required.  
**modifier-state:** An instance of type **<integer>**. Required.  
**modifiers:** An instance of type **<sequence>**.

Description The base class of all gestures.

Operations **add-command** **add-command-table-menu-item** **event-matches-gesture?** **gadget-accelerator-setter** **gesture-modifier-state** **gesture-spec-equal**

See also [<keyboard-gesture>](#), page 255

[<pointer-gesture>](#), page 285

## **gesture-button** *Generic function*

Summary Returns the button associated with the specified gesture.

Signature **gesture-button** **pointer-gesture => button**

Arguments **pointer-gesture** An instance of type **<pointer-gesture>**.

Values **button** An instance of type **<integer>**.

Description      Returns the button associated with *pointer-gesture*.

See also      `<pointer-gesture>`, page 285

## gesture-keysym

*Generic function*

Summary      Returns the keysym associated with the specified gesture.

Signature      `gesture-keysym keyboard-gesture => keysym`

Arguments      *keyboard-gesture* An instance of type `<keyboard-gesture>`.

Values      *keysym*      An instance of type `<symbol>`.

Description      Returns the keysym associated with *keyboard-gesture*.

See also      `<keyboard-gesture>`, page 255

## gesture-modifier-state

*Generic function*

Summary      Returns the modifier-state associated with the specified gesture.

Signature      `gesture-modifier-state gesture => modifier-state`

Arguments      *gesture*      An instance of type `<gesture>`.

Values      *modifier-state*      An instance of type `<integer>`.

Description      Returns the modifier-state associated with *gesture*.

See also      `event-modifier-state`, page 234

`<keyboard-gesture>`, page 255

`make-modifier-state`, page 258

`port-modifier-state`, page 289

## gesture-spec-equal *Function*

Summary	Returns true if the two specified gestures are equivalent.
Signature	<code>gesture-spec-equal gesture1 gesture2 =&gt; equal?</code>
Arguments	<code>gesture1</code> An instance of type <code>&lt;gesture&gt;</code> . <code>gesture2</code> An instance of type <code>&lt;gesture&gt;</code> .
Values	<code>equal?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <code>gesture1</code> and <code>gesture2</code> are equivalent.
See also	=, page 183

## get-clipboard-data-as *Generic function*

Summary	Returns data of a given type from a clipboard.
Signature	<code>get-clipboard-data-as type clipboard =&gt; data</code>
Arguments	<code>type</code> An instance of <code>type-union(&lt;symbol&gt;, &lt;type&gt;)</code> . <code>clipboard</code> An instance of <code>&lt;clipboard&gt;</code> .
Values	<code>data</code> Instances of <code>&lt;object&gt;</code> .
Description	This generic function returns <code>data</code> of <code>type</code> from the clipboard. The argument <code>type</code> is an instance of <code>type-union(&lt;symbol&gt;, &lt;type&gt;)</code> .
See also	<code>add-clipboard-data</code> , page 184

**add-clipboard-data-as**, page 185  
**<clipboard>**, page 208  
**clipboard-data-available?**, page 209

## get-default-background *Generic function*

Summary	Returns the default background for the specified sheet.	
Signature	<b>get-default-background</b> <i>port sheet</i> #key <b>background</b> => <i>background</i>	
Arguments	<i>port</i>	An instance of type <b>&lt;port&gt;</b> .
	<i>sheet</i>	An instance of type <b>&lt;sheets&gt;</b> .
	<i>background</i>	An instance of type <b>&lt;ink&gt;</b> .
Values	<i>background</i>	An instance of type <b>&lt;ink&gt;</b> .
Description	Returns the default background for <i>sheet</i> on <i>port</i> . If <i>background</i> is specified, then this is used instead of the default.	
See also	<b>get-default-foreground</b> , page 250 <b>get-default-text-style</b> , page 251	

## get-default-foreground *Generic function*

Summary	Returns the default foreground for the specified sheet.	
Signature	<b>get-default-foreground</b> <i>port sheet</i> #key <b>foreground</b> => <i>foreground</i>	
Arguments	<i>port</i>	An instance of type <b>&lt;port&gt;</b> .

	<i>sheet</i>	An instance of type <sheet>.
	<i>foreground</i>	An instance of type <ink>.
Values	<i>foreground</i>	An instance of type <ink>.
Description		Returns the default foreground for <i>sheet</i> on <i>port</i> . If <i>foreground</i> is specified, then this is used instead of the default.
See also		<code>get-default-background</code> , page 250 <code>get-default-text-style</code> , page 251

## get-default-text-style *Generic function*

Summary	Returns the default text style for the specified sheet.	
Signature	<code>get-default-text-style port sheet #key text-style =&gt; text-style</code>	
Arguments	<i>port</i>	An instance of type <port>.
	<i>sheet</i>	An instance of type <sheet>.
	<i>text-style</i>	An instance of type <text-style>.
Values	<i>text-style</i>	An instance of type <text-style>.
Description		Returns the default text style for <i>sheet</i> on <i>port</i> . If <i>text-style</i> is specified, then this is used instead of the default.
See also		<code>get-default-background</code> , page 250 <code>get-default-foreground</code> , page 250

	<i>Generic function</i>				
<b>handle-event</b>					
Summary	Implements any defined policies of the specified sheet with respect to the specified event.				
Signature	<code>handle-event sheet event =&gt; ()</code>				
Arguments	<table> <tr> <td><i>sheet</i></td><td>An instance of type <code>&lt;sheet&gt;</code>.</td></tr> <tr> <td><i>event</i></td><td>An instance of type <code>&lt;event&gt;</code>.</td></tr> </table>	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .	<i>event</i>	An instance of type <code>&lt;event&gt;</code> .
<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .				
<i>event</i>	An instance of type <code>&lt;event&gt;</code> .				
Values	None				
Description	<p>Implements any defined policies of <i>sheet</i> with respect to <i>event</i>. Methods defined on this generic are called by DUIM to do the handling.</p> <p>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.</p> <p>DUIM itself implements no semantically meaningful <code>handle-event</code> methods; It is the responsibility of any application to implement all of its own <code>handle-event</code> methods. It is also the responsibility of the application to decide the protocol and relationship between all of these methods.</p> <p>Take care when adding <code>next-method()</code> calls in any <code>handle-event</code> 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 <code>next-method</code> is actually required.</p>				
See also	<p><code>handle-repaint</code>, page 253</p> <p><code>queue-event</code>, page 292</p>				

## handle-repaint *Generic function*

Summary	Implements region repainting for a given sheet class.						
Signature	<code>handle-repaint <i>sheet</i> <i>medium</i> <i>region</i> =&gt; ()</code>						
Arguments	<table border="0"> <tr> <td><i>sheet</i></td><td>An instance of type &lt;<i>sheet</i>&gt;.</td></tr> <tr> <td><i>medium</i></td><td>An instance of type &lt;<i>medium</i>&gt;.</td></tr> <tr> <td><i>region</i></td><td>An instance of type &lt;<i>region</i>&gt;.</td></tr> </table>	<i>sheet</i>	An instance of type < <i>sheet</i> >.	<i>medium</i>	An instance of type < <i>medium</i> >.	<i>region</i>	An instance of type < <i>region</i> >.
<i>sheet</i>	An instance of type < <i>sheet</i> >.						
<i>medium</i>	An instance of type < <i>medium</i> >.						
<i>region</i>	An instance of type < <i>region</i> >.						
Values	None						
Description	<p>Implements region repainting for a given sheet class. Methods on this generic are 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 <i>region</i> of the <i>sheet</i> on <i>medium</i>.</p> <p>DUIM itself implements no semantically meaningful <code>handle-repaint</code> methods; It is the responsibility of any application to implement all of its own <code>handle-repaint</code> methods. It is also the responsibility of the application to decide the protocol and relationship between all of these methods.</p> <p>Take care when adding <code>next-method()</code> calls in any <code>handle-repaint</code> 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 <code>next-method</code> is actually required.</p> <p>The <i>sheet</i> on <i>medium</i> is repainted and <i>region</i> is the region to repaint.</p>						
See also	<p><code>&lt;drawing-pane&gt;</code>, page 409</p> <p><code>pane-display-function</code>, page 420</p> <p><code>queue-repaint</code>, page 293</p>						

`repaint-sheet`, page 294  
`<simple-pane>`, page 426  
`<window-repaint-event>`, page 326

## \$hyper-key *Constant*

Summary A constant that represents the HYPER key on the keyboard.

Type `<integer>`

Value `ash(1, %modifier_base + 4);`

Description A constant that represents the HYPER key on the keyboard.

See also `$alt-key`, page 185

`$control-key`, page 211

`$meta-key`, page 271

`modifier-key-index`, page 273

`modifier-key-index-name`, page 273

`$modifier-keys`, page 274

`$option-key`, page 278

`$shift-key`, page 317

`$super-key`, page 318

## <keyboard-event> *Open abstract class*

Summary The base class of all keyboard events.

Superclasses `<device-event>`

Init-keywords	<b>key-name:</b> An instance of type <code>false-or(&lt;symbol&gt;)</code> . Default value: <code>#f</code> .
	<b>character:</b> An instance of type <code>false-or(&lt;character&gt;)</code> . Default value: <code>#f</code> .
Description	The base class of all keyboard events.
	The key-name: init-keyword represents the name of the key on the keyboard that was pressed.
	The <b>character:</b> init-keyword represents the keyboard character that was pressed for characters in the standard character set.
Operations	The following operations are exported from the <code>DUIM-Sheets</code> module.
	<code>event-character event-key-name event-matches-gesture?</code>
See also	<code>event-character</code> , page 232
	<code>event-key-name</code> , page 233
	<code>&lt;key-press-event&gt;</code> , page 256
	<code>&lt;key-release-event&gt;</code> , page 256

## <keyboard-gesture>

*Sealed instantiable class*

Summary	The base class of all keyboard gestures.
Superclasses	<gesture>
Init-keywords	<b>keysym:</b> An instance of type <code>&lt;symbol&gt;</code> . <b>modifier-state:</b> An instance of type <code>&lt;integer&gt;</code> .
Description	The base class of all keyboard gestures.

The `keysym`: init-keyword represents the keysym for the gesture, and the `modifier-state`: init-keyword represents its modifier state.

Operations      `gesture-keysym`

See also      `gesture-keysym`, page 248

`gesture-modifier-state`, page 248

## **<key-press-event>**

*Sealed instantiable class*

Summary      The class of events passed when a key is pressed.

Superclasses    `<keyboard-event>`

Init-keywords   None.

Description      The class of events passed when a key is pressed.

Operations      None.

See also      `<keyboard-event>`, page 254

`<key-release-event>`, page 256

## **<key-release-event>**

*Sealed instantiable class*

Summary      The class of events passed when a key is released.

Superclasses    `<keyboard-event>`

Init-keywords   None.

Description      The class of events passed when a key is released.

Operations	None.
See also	<a href="#"><code>&lt;keyboard-event&gt;</code>, page 254</a> <a href="#"><code>&lt;key-press-event&gt;</code>, page 256</a>

## \$left-button *Constant*

Summary	A constant that represents the left button on the attached pointing device.
Type	<code>&lt;integer&gt;</code>
Value	<code>ash(1, %button_base + 0)</code>
Description	A constant that represents the left button on the attached pointing device.
See also	<a href="#"><code>\$middle-button</code>, page 272</a> <a href="#"><code>\$pointer-buttons</code>, page 281</a> <a href="#"><code>\$right-button</code>, page 295</a>

## lower-sheet *Generic function*

Summary	Lowers the specified sheet to the bottom of the current hierarchy of sheets.		
Signature	<code>lower-sheet <i>sheet</i> =&gt; ()</code>		
Arguments	<table> <tr> <td><i>sheet</i></td> <td>An instance of type <code>&lt;sheet&gt;</code>.</td> </tr> </table>	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .		
Values	None		

Description      Lowers *sheet* to the bottom of the current hierarchy of sheets.

See also      **`lower-frame`**, page 727  
**`raise-frame`**, page 741  
**`raise-sheet`**, page 293

## make-frame-manager *Generic function*

Summary      Returns an instance of **<frame-manager>** on the specified port.

Signature      **`make-frame-manager port #key palette => framem`**

Arguments      ***port***      An instance of type **<port>**.  
***palette***      An instance of type **<palette>**.

Values      ***framem***      An instance of type **<frame-manager>**.

Description      Returns an instance of **<frame-manager>** on *port*. If specified, the palette described by *palette* is used.

See also      **<frame-manager>**, page 243

## make-modifier-state *Function*

Summary      Returns a modifier state for the specified modifiers.

Signature      **`make-modifier-state #rest modifiers => integer`**

Arguments      ***modifiers***      An instance of type **limited(<sequence>, of: <integer>)**.

Values      ***integer***      An instance of type **<integer>**.

Description      Returns a modifier state for *modifiers*.

See also [event-modifier-state](#), page 234  
[gesture-modifier-state](#), page 248  
[port-modifier-state](#), page 289

## **make-pane** *Generic function*

Summary Selects and returns an instance of a suitable class of pane for the supplied options.

Signature **make-pane pane-class #rest pane-options #key frame-manager => sheet**

Arguments **pane-class** An instance of type **<class>**.  
**pane-options** Instances of type **<object>**.  
**frame-manager** An instance of type **<frame-manager>**.

Values **sheet** An instance of type **<sheet>**.

Description Selects a class that implements the behavior of pane-class and constructs a pane of that class.

## **<medium>** *Open abstract instantiable class*

Summary The class of all mediums.

Superclasses **<object>**

Init-keywords None.

Description The class of all mediums.

Mediums have the following elements associated with them:

- A drawing plane, to which text and lines may be drawn

- A foreground color, which describes the default color of anything drawn on the drawing plane
- A background color, which describes the background color of the drawing plane
- A transformation which describes the position of the drawing plane relative to the sheet which is its parent
- A clipping region, on which any editing operations (such as cutting, copying, or pasting) will have effect.
- A line style that describes the appearance of any lines drawn on the drawing plane
- A text style that describes the appearance of any text written to the drawing plane

**Operations** The following operations are exported from the `DUIM-Sheets` module.

```
beep clear-box display do-with-drawing-options do-
with-text-style do-with-transform force-display
handle-repaint medium? medium-background medium-back-
ground-setter medium-brush medium-brush-setter
medium-clipping-region medium-clipping-region-setter
medium-default-text-style medium-default-text-style-
setter medium-drawable medium-drawable-setter medium-
foreground medium-foreground-setter medium-merged-
text-style medium-pen medium-pen-setter medium-pixmap
medium-pixmap-setter medium-sheet medium-text-style
medium-text-style-setter medium-transform medium-
transform-setter portsynchronize-display text-size
```

The following operations are exported from the `DUIM-Graphics` module.

```
copy-area copy-from-pixmap copy-to-pixmap do-with-
output-to-pixmap draw-bezier-curve draw-image make-
pixmap
```

The following operations are exported from the **DUIM-Extended-Geometry** module.

**draw-design**

See also      [medium?](#), page 261  
[<pixmap-medium>](#), page 391

## medium?

*Generic function*

Summary     Returns true if the specified object is a medium.

Signature    **medium? object => medium?**

Arguments    *object*       An instance of type [<object>](#).

Values       *medium?*      An instance of type [<boolean>](#).

Description    Returns true if *object* is a medium.

See also     [<medium>](#), page 259  
[sheet?](#), page 303

## medium-background

*Generic function*

Summary     Returns the background for the specified medium.

Signature    **medium-background medium => ink**

Arguments    *medium*       An instance of type [<medium>](#).

Values       *ink*          An instance of type [<ink>](#).

Description    Returns the background for *medium*.

See also      [medium-background-setter, page 262](#)  
[medium-foreground, page 266](#)

## medium-background-setter                          *Generic function*

Summary      Sets the background for the specified medium.

Signature      `medium-background-setter background medium => background`

Arguments      `background`      An instance of type `<ink>`.  
                  `medium`      An instance of type `<medium>`.

Values      `background`      An instance of type `<ink>`.

Description      Sets the background for `medium`.

See also      [medium-background, page 261](#)  
[medium-foreground-setter, page 266](#)

## medium-brush                          *Generic function*

Summary      Returns the brush for the specified medium.

Signature      `medium-brush medium => brush`

Arguments      `medium`      An instance of type `<medium>`.

Values      `brush`      An instance of type `<brush>`.

Description      Returns the brush for `medium`. This brush is used by all subsequent painting operations on `medium`.

See also      [medium-brush-setter, page 263](#)

`medium-pen`, page 267

## medium-brush-setter

*Generic function*

Summary	Sets the brush for the specified medium.	
Signature	<code>medium-brush-setter brush medium =&gt; brush</code>	
Arguments	<i>brush</i>	An instance of type <code>&lt;brush&gt;</code> .
	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>brush</i>	An instance of type <code>&lt;brush&gt;</code> .
Description	Sets the brush for <i>medium</i> . This brush is used by all subsequent painting operations on <i>medium</i> .	
See also	<a href="#">medium-brush</a> , page 262 <a href="#">medium-pen-setter</a> , page 268	

## medium-clipping-region

*Generic function*

Summary	Returns the clipping region for the specified medium.	
Signature	<code>medium-clipping-region medium =&gt; region</code>	
Arguments	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>region</i>	An instance of type <code>&lt;region&gt;</code> .
Description	Returns the clipping region for <i>medium</i> .	
See also	<a href="#">medium-clipping-region-setter</a> , page 264	

## medium-clipping-region-setter *Generic function*

Summary      Sets the clipping region for the specified medium.

Signature      `medium-clipping-region-setter region medium => region`

Arguments      `region`      An instance of type `<region>`.  
                  `medium`      An instance of type `<medium>`.

Values      `region`      An instance of type `<region>`.

Description      Sets the clipping region for *medium*.

See also      `medium-clipping-region`, page 263

## medium-default-text-style *Generic function*

Summary      Returns the default text style for the specified medium.

Signature      `medium-default-text-style medium => text-style`

Arguments      `medium`      An instance of type `<medium>`.

Values      `text-style`      An instance of type `<text-style>`.

Description      Returns the default text style for *medium*. This style is used for any subsequent text that is written to *medium*.

See also      `medium-default-text-style-setter`, page 265

`medium-merged-text-style`, page 267

`medium-text-style`, page 270

**medium-default-text-style-setter** *Generic function*

Summary	Sets the default text style for the specified medium.	
Signature	<code>medium-default-text-style-setter text-style medium =&gt; text-style</code>	
Arguments	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Description	Sets the default text style for <i>medium</i> . This style is used for any subsequent text that is written to <i>medium</i> .	
See also	<a href="#">medium-default-text-style</a> , page 264 <a href="#">medium-text-style-setter</a> , page 270	

**medium-drawable** *Generic function*

Summary	Returns the drawable for the specified medium.	
Signature	<code>medium-drawable medium =&gt; drawable</code>	
Arguments	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>drawable</i>	An instance of type <code>&lt;object&gt;</code> .
Description	Returns the drawable for <i>medium</i> .	
See also	<a href="#">medium-drawable-setter</a> , page 266	

## medium-drawable-setter *Generic function*

Summary	Sets the drawable for the specified medium.	
Signature	<code>medium-drawable-setter drawable medium =&gt; object</code>	
Arguments	<code>drawable</code>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	<code>medium</code>	An instance of type <code>&lt;medium&gt;</code> .
Values	<code>object</code> An instance of type <code>&lt;object&gt;</code> .	
Description	Sets the drawable for <i>medium</i> .	
See also	<a href="#">medium-drawable</a> , page 265	

## medium-foreground *Generic function*

Summary	Returns the foreground of the specified medium.	
Signature	<code>medium-foreground medium =&gt; ink</code>	
Arguments	<code>medium</code>	An instance of type <code>&lt;medium&gt;</code> .
Values	<code>ink</code>	An instance of type <code>&lt;ink&gt;</code> .
Description	Returns the foreground of <i>medium</i> .	
See also	<a href="#">medium-background</a> , page 261 <a href="#">medium-foreground-setter</a> , page 266	

## medium-foreground-setter *Generic function*

Summary	Sets the foreground of the specified medium.
---------	--

Signature	<code>medium-foreground-setter</code> <i>foreground</i> <i>medium</i> => <i>foreground</i>	
Arguments	<i>foreground</i>	An instance of type <code>&lt;ink&gt;</code> .
	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>foreground</i>	An instance of type <code>&lt;ink&gt;</code> .
Description	Sets the foreground of <i>medium</i> .	
See also	<a href="#">medium-background-setter</a> , page 262 <a href="#">medium-foreground</a> , page 266	

## **medium-merged-text-style** *Generic function*

Summary	Returns the merged text style of the specified medium.	
Signature	<code>medium-merged-text-style</code> <i>medium</i> => <i>text-style</i>	
Arguments	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
Description	Returns the merged text style of <i>medium</i> .	
See also	<a href="#">medium-default-text-style</a> , page 264 <a href="#">medium-text-style</a> , page 270	

## **medium-pen** *Generic function*

Summary	Returns the pen for the specified medium.	
Signature	<code>medium-pen</code> <i>medium</i> => <i>pen</i>	
Arguments	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .

Values	<i>pen</i>	An instance of type < <b>pen</b> >.
Description	Returns the pen for <i>medium</i> . This brush is used by all subsequent drawing operations on <i>medium</i> .	
See also	<a href="#">medium-brush</a> , page 262 <a href="#">medium-pen-setter</a> , page 268	

## **medium-pen-setter** *Generic function*

Summary	Sets the pen for the specified medium.	
Signature	<b>medium-pen-setter</b> <i>pen medium =&gt; pen</i>	
Arguments	<i>pen</i>	An instance of type < <b>pen</b> >.
	<i>medium</i>	An instance of type < <b>medium</b> >.
Values	<i>pen</i>	An instance of type < <b>pen</b> >.
Description	Sets the pen for <i>medium</i> . This brush is used by all subsequent drawing operations on <i>medium</i> .	
See also	<a href="#">medium-brush-setter</a> , page 263 <a href="#">medium-pen</a> , page 267	

## **medium-pixmap** *Generic function*

Summary	Returns the pixmap for the specified medium.	
Signature	<b>medium-pixmap</b> <i>medium =&gt; value</i>	
Arguments	<i>medium</i>	An instance of type < <b>medium</b> >.

Values	<i>value</i>	An instance of type <code>false-or(&lt;pixmap&gt;)</code> .
Description	Returns the pixmap for <i>medium</i> . This pixmap is used by all subsequent pixmap operations on <i>medium</i> .	
See also		<a href="#">medium-pixmap-setter</a> , page 269

## medium-pixmap-setter *Generic function*

Summary	Sets the pixmap for the specified medium.	
Signature	<code>medium-pixmap-setter pixmap medium =&gt; value</code>	
Arguments	<i>pixmap</i>	An instance of type <code>&lt;pixmap&gt;</code> .
	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>value</i>	An instance of type <code>false-or(&lt;pixmap&gt;)</code> .
Description	Returns the pixmap for <i>medium</i> . This pixmap is used by all subsequent pixmap operations on <i>medium</i> .	
See also	<a href="#">medium-pixmap</a> , page 268	

## medium-sheet *Generic function*

Summary	Returns the sheet for the specified medium.	
Signature	<code>medium-sheet medium =&gt; sheet</code>	
Arguments	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	<i>sheet</i>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Returns the sheet for <i>medium</i> , if there is one.	

## medium-text-style *Generic function*

Summary	Returns the text style for the specified medium.	
Signature	<code>medium-text-style <b>medium</b> =&gt; <b>text-style</b></code>	
Arguments	<b>medium</b>	An instance of type < <code>medium</code> >.
Values	<b>text-style</b>	An instance of type < <code>text-style</code> >.
Description	Returns the text style for <i>medium</i> .	
See also	<a href="#">medium-default-text-style, page 264</a> <a href="#">medium-merged-text-style, page 267</a> <a href="#">medium-text-style-setter, page 270</a>	

## medium-text-style-setter *Generic function*

Summary	Sets the text style for the specified medium.	
Signature	<code>medium-text-style-setter <b>text-style</b> <b>medium</b> =&gt; <b>text-style</b></code>	
Arguments	<b>text-style</b>	An instance of type < <code>text-style</code> >.
	<b>medium</b>	An instance of type < <code>medium</code> >.
Values	<b>text-style</b>	An instance of type < <code>text-style</code> >.
Description	Sets the text style for <i>medium</i> .	
See also	<a href="#">medium-default-text-style-setter, page 265</a> <a href="#">medium-text-style, page 270</a>	

## **medium-transform** *Generic function*

Summary	Returns the transform for the specified medium.	
Signature	<code>medium-transform medium =&gt; transform</code>	
Arguments	<i>medium</i>	An instance of type < <code>medium</code> >.
Values	<i>transform</i>	An instance of type < <code>transform</code> >.
Description	Returns the transform for <i>medium</i> .	
See also	<a href="#">medium-transform-setter</a> , page 271 <a href="#">sheet-transform</a> , page 316	

## **medium-transform-setter** *Generic function*

Summary	Sets the transform for the specified medium.	
Signature	<code>medium-transform-setter transform medium =&gt; transform</code>	
Arguments	<i>transform</i>	An instance of type < <code>transform</code> >.
	<i>medium</i>	An instance of type < <code>medium</code> >.
Values	<i>transform</i>	An instance of type < <code>transform</code> >.
Description	Sets the transform for <i>medium</i> .	
See also	<a href="#">medium-transform</a> , page 271 <a href="#">sheet-transform-setter</a> , page 317	

## **\$meta-key** *Constant*

Summary	A constant that represents the META key on the keyboard.
---------	--

Type	<code>&lt;integer&gt;</code>
Value	<code>ash(1, %modifier_base + 2);</code>
Description	A constant that represents the META key on the keyboard, if it exists. To deal with the case where there is no META key, the value of the constant <code>\$alt-key</code> is bound to this constant.
See also	<code>\$alt-key</code> , page 185 <code>\$control-key</code> , page 211 <code>\$hyper-key</code> , page 254 <code>modifier-key-index</code> , page 273 <code>modifier-key-index-name</code> , page 273 <code>\$modifier-keys</code> , page 274 <code>\$option-key</code> , page 278 <code>\$shift-key</code> , page 317 <code>\$super-key</code> , page 318

## **\$middle-button**

*Constant*

Summary	A constant that represents the middle button on the attached pointing device.
Type	<code>&lt;integer&gt;</code>
Value	<code>ash(1, %button_base + 1)</code>
Description	A constant that represents the middle button on the attached pointing device.
See also	<code>\$left-button</code> , page 257 <code>\$pointer-buttons</code> , page 281

`$right-button`, page 295

## modifier-key-index *Function*

Summary	Returns the index number of the specified modifier key.
Signature	<code>modifier-key-index key-name =&gt; index</code>
Arguments	<code>key-name</code> An instance of type <code>&lt;symbol&gt;</code> .
Values	<code>index</code> An instance of type <code>&lt;integer&gt;</code> .
Description	<p>Returns the index number of the specified modifier key. The <code>key-name</code> specified may be any of the elements of <code>\$modifier-keys</code></p> <p>The returned index value is either 0, 1, 2, 3, or 4.</p>
See also	<p><code>\$alt-key</code>, page 185</p> <p><code>\$control-key</code>, page 211</p> <p><code>\$hyper-key</code>, page 254</p> <p><code>\$meta-key</code>, page 271</p> <p><code>modifier-key-index-name</code>, page 273</p> <p><code>\$modifier-keys</code>, page 274</p> <p><code>\$option-key</code>, page 278</p> <p><code>\$shift-key</code>, page 317</p> <p><code>\$super-key</code>, page 318</p>

## modifier-key-index-name *Function*

Summary	Returns the key name of the specified modifier key index.
---------	---

Signature	<code>modifier-key-index-name index =&gt; key-name</code>	
Arguments	<code>index</code>	An instance of type <code>&lt;integer&gt;</code> .
Values	<code>key-name</code>	An instance of type <code>&lt;symbol&gt;</code> .
Description	<p>Returns the key name of the specified modifier key index.  The <code>index</code> specified is either 0, 1, 2, 3, or 4.</p> <p>The <code>key-name</code> returned may be any of the elements of  <code>\$modifier-keys</code></p>	
See also	<p><code>\$alt-key</code>, page 185</p> <p><code>\$control-key</code>, page 211</p> <p><code>\$hyper-key</code>, page 254</p> <p><code>\$meta-key</code>, page 271</p> <p><code>modifier-key-index</code>, page 273</p> <p><code>\$modifier-keys</code>, page 274</p> <p><code>\$option-key</code>, page 278</p> <p><code>\$shift-key</code>, page 317</p> <p><code>\$super-key</code>, page 318</p>	

## \$modifier-keys *Constant*

Summary	The default list of keys on the keyboard that are used as modifiers.
Type	<code>&lt;sequence&gt;</code>
Value	<code>#[#"shift", #"control", #"meta", #"super", #"hyper"]</code>
Description	The default list of keys on the keyboard that are used as modifiers for keyboard accelerators and mnemonics.

See also [\\$alt-key](#), page 185  
[\\$control-key](#), page 211  
[\\$hyper-key](#), page 254  
[\\$meta-key](#), page 271  
[modifier-key-index](#), page 273  
[modifier-key-index-name](#), page 273  
[\\$option-key](#), page 278  
[\\$shift-key](#), page 317  
[\\$super-key](#), page 318

	<b>notify-user</b>	<i>Generic function</i>
Summary		Creates and displays an alert dialog box with the specified criteria.
Signature		<b>notify-user message-string #key frame owner title documentation exit-boxes name style foreground background text-style =&gt; boolean</b>
Arguments	<i>message-string</i>	An instance of type <string>.
	<i>frame</i>	An instance of type <frame>. Default value: <code>current-frame()</code> .
	<i>owner</i>	An instance of type <sheet>.
	<i>title</i>	An instance of type <string>.
	<i>documentation</i>	An instance of type <code>false-or(&lt;string&gt;)</code> . Default value: <code>#f</code> .
	<i>exit-boxes</i>	An instance of type <object>.
	<i>name</i>	An instance of type <object>.

	<i>style</i>	An instance of type <code>one-of(#"information", #"question", #"warning", #"error", #"serious-error", #"fatal-error")</code> .
	<i>foreground</i>	An instance of type <code>false-or(&lt;ink&gt;)</code> . Default value: <code>#f</code> .
	<i>background</i>	An instance of type <code>false-or(&lt;ink&gt;)</code> . Default value: <code>#f</code> .
	<i>text-style</i>	An instance of type <code>false-or(&lt;text-style&gt;)</code> . Default value: <code>#f</code> .
Values	<i>boolean</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description		Creates and displays an alert dialog box with the specified criteria. Use this function as a way of easily displaying simple messages to the user.



**Figure 5.3** Simple output from `notify-user`

The *message-string* is the message that is displayed in the dialog. The arguments frame, owner, title, and documentation let you specify different attributes for the dialog in the same way as they can be specified for any other frame or dialog.

The *exit-boxes* argument lets you specify the buttons that are available in the dialog. If not supplied, then a single **OK** button is used by default, unless the *style* of the dialog is set to `#"question"`, in which case, two buttons are created, to allow the user to respond “yes” or “no”.

The *style* argument lets you specify the style of dialog that is produced. The different styles available reflect the Motif specification for dialog box types. Depending on the style of dialog you choose, the appearance of the dialog created may vary. For example, a different icon is commonly used to distinguish between error, informational, and warning messages.

The *foreground*, *background*, and *text-style* arguments let you specify foreground and background colors, and the font to use in the message text.

See also [choose-color](#), page 194  
[choose-directory](#), page 196  
[choose-file](#), page 198

		<i>Function</i>
Summary		Creates a clipboard lock for a sheet on a port.
Signature		<b>open-clipboard</b> <i>port sheet</i> => <i>clipboard</i>
Arguments	<i>port</i>	An instance of < <i>port</i> >.
	<i>sheet</i>	An instance of < <i>sheet</i> >.
Values	<i>clipboard</i>	An instance of < <i>clipboard</i> >.
Description		Creates a clipboard lock for <i>sheet</i> on <i>port</i> . Once a clipboard lock has been created, you can manipulate the clipboard contents safely. An instance of < <i>clipboard</i> > is returned, which is used to hold the clipboard contents.  You should not normally call <b>open-clipboard</b> yourself to create a clipboard lock. Use the macro <b>with-clipboard</b> to create and free the lock for you.

See also      [`<clipboard>`](#), page 208  
[`with-clipboard`](#), page 327

## **\$option-key** *Constant*

Summary      A constant that represents the OPTION key on the keyboard.

Type      [`<integer>`](#)

Value      [`\$super-key`](#)

Description      A constant that represents the OPTION key on the keyboard.  
This is set to the same value as the SUPER key, to deal with  
the case where the OPTION key is not present on the key-  
board.

See also      [`\$alt-key`](#), page 185  
[`\$control-key`](#), page 211  
[`\$hyper-key`](#), page 254  
[`\$meta-key`](#), page 271  
[`modifier-key-index`](#), page 273  
[`modifier-key-index-name`](#), page 273  
[`\$modifier-keys`](#), page 274  
[`\$shift-key`](#), page 317  
[`\$super-key`](#), page 318

## **<pointer>** *Open abstract instantiable class*

Summary      The class of all pointers.

Superclasses      [`<object>`](#)

Init-keywords	<code>port:</code>	An instance of type <code>&lt;port&gt;</code> .
Description	The class of all pointers.	
Operations	The following operations are exported from the <code>DUIM-Sheets</code> module.	
		<code>display pointer? pointer-button-state pointer-cursor pointer-cursor-setter pointer-position pointer-sheet port set-pointer-position</code>

See also      `pointer?`, page 279

## **pointer?** *Generic function*

Summary	Returns true if the specified object is a pointer.	
Signature	<code>pointer? object =&gt; pointer?</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>pointer?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>object</i> is a pointer.	
See also	<code>&lt;pointer&gt;</code> , page 278	

## **<pointer-boundary-event>** *Sealed instantiable class*

Summary	The class that corresponds to a pointer motion event that crosses a sheet boundary.	
Superclasses	<code>&lt;pointer-motion-event&gt;</code>	

Init-keywords	<b>kind:</b>	An instance of type <code>one-of(#"ancestor", "#virtual", #"inferior", #"nonlinear", #"nonlinear-virtual", #f)</code> . Default value: <code>#f</code> .
Description	The class that corresponds to a pointer motion event that crosses some sort of sheet boundary.	The <code>kind:</code> init-keyword represents the boundary event kind. These correspond to the detail members for X11 enter and exit events.
Operations	The following operation is exported from the <code>DUIM-Sheets</code> module.	<code>boundary-event-kind</code>
See also	<code>boundary-event-kind</code> , page 186 <code>&lt;pointer-enter-event&gt;</code> , page 283 <code>&lt;pointer-exit-event&gt;</code> , page 284	

**<pointer-button-event>***Open abstract class*

Summary	The class of events that occur when mouse buttons are pressed.
Superclasses	<code>&lt;pointer-event&gt;</code>
Init-keywords	<b>button:</b> An instance of type <code>one-of(\$left-button, \$middle-button, \$right-button)</code> .
Description	The class of events that occur when mouse buttons are pressed.
Operations	The following operations are exported from the <code>DUIM-Sheets</code> module.

**event-button event-matches-gesture? handle-event**

See also [event-button](#), page 231  
[\\$left-button](#), page 257  
[\\$middle-button](#), page 272  
[pointer-button-state](#), page 282  
[<pointer-drag-event>](#), page 283  
[\\$right-button](#), page 295

## \$pointer-buttons *Constant*

Summary The constant representing the possible buttons on the pointing device.

Type `<sequence>`

Value `#["left", "middle", "right"];`

Description The constant representing the possible buttons on the pointing device attached to the computer, typically a mouse. Up to three buttons are provided for.  
The order of the elements in this sequence must match the order of the values of `$left-button`, `$middle-button`, and `$right-button`

See also [button-index](#), page 187  
[button-index-name](#), page 187  
[\\$left-button](#), page 257  
[\\$middle-button](#), page 272  
[\\$right-button](#), page 295

**pointer-button-state** *Generic function*

Summary      Returns the state of the specified pointer.

Signature     `pointer-button-state pointer => integer`

Arguments    `pointer`      An instance of type `<pointer>`.

Values        `integer`      An instance of type `<integer>`.

Description    Returns the state of *pointer*.

**pointer-cursor** *Generic function*

Summary      Returns the cursor used for the specified pointer.

Signature     `pointer-cursor pointer => cursor`

Arguments    `pointer`      An instance of type `<pointer>`.

Values        `cursor`      An instance of type `<cursor>`.

Description    Returns the cursor used for *pointer*.

See also      `pointer-cursor-setter`, page 282

**pointer-cursor-setter** *Generic function*

Summary      Sets the cursor used for the specified pointer.

Signature     `pointer-cursor-setter cursor pointer => cursor`

Arguments    `cursor`      An instance of type `<cursor>`.

`pointer`      An instance of type `<pointer>`.

Values	<code>cursor</code>	An instance of type <code>&lt;cursor&gt;</code> .
Description		Sets the cursor used for <i>pointer</i> .
See also		<code>pointer-cursor</code> , page 282

## `<pointer-drag-event>` *Sealed instantiable class*

Summary	The class of events describing drag movements.
Superclasses	<code>&lt;pointer-motion-event&gt;</code> <code>&lt;pointer-button-event&gt;</code>
Init-keywords	<code>button:</code> An instance of type <code>one-of(\$left-button, \$middle-button, \$right-button)</code> .
Description	The class of events describing drag movements. This is the same as <code>&lt;pointer-motion-event&gt;</code> , except that a button on the attached pointing device must also be held down as the pointer is moving. The <code>button:</code> init-keyword is inherited from the superclass <code>&lt;pointer-button-event&gt;</code> .
Operations	None.
See also	<code>&lt;pointer-motion-event&gt;</code> , page 285

## `<pointer-enter-event>` *Sealed instantiable class*

Summary	The class of events that describe a pointer entering an area such as a sheet.
Superclasses	<code>&lt;pointer-boundary-event&gt;</code>
Init-keywords	None.

Description      The class of events that describe a pointer entering an area such as a sheet.

Operations      None.

See also      [`<pointer-exit-event>`](#), page 284

## **<pointer-event>**

*Open abstract class*

Summary      The base class of events occurring on pointers.

Superclasses      [`<device-event>`](#)

Init-keywords      **x:**      An instance of type [`<real>`](#).  
**y:**      An instance of type [`<real>`](#).  
**pointer:**      An instance of type [`<pointer>`](#).

Description      The base class of events occurring on pointers on the computer screen.

The **x:** and **y:** init-keywords specify the location of the pointer when the event occurs. The **pointer:** init-keyword specifies the pointer to which the event occurs.

Operations      None.

See also      [`<pointer-button-event>`](#), page 280

[`<pointer-exit-event>`](#), page 284

[`<pointer-motion-event>`](#), page 285

## **<pointer-exit-event>**

*Sealed instantiable class*

Summary      The class of events that describe a pointer leaving an area such as a sheet.

Superclasses	<code>&lt;pointer-boundary-event&gt;</code>
Init-keywords	None.
Description	The class of events that describe a pointer leaving an area such as a sheet.
Operations	None.
See also	<code>&lt;pointer-button-event&gt;</code> , page 280 <code>&lt;pointer-enter-event&gt;</code> , page 283 <code>&lt;pointer-motion-event&gt;</code> , page 285

**<pointer-gesture>***Sealed instantiable class*

Summary	The class of all gestures that occur on pointers.
Superclasses	<code>&lt;gesture&gt;</code>
Init-keywords	<code>button:</code> An instance of type <code>&lt;integer&gt;</code> . <code>modifier-state:</code> An instance of type <code>&lt;integer&gt;</code> .
Description	<p>The class of all gestures that occur on pointers.</p> <p>The <code>button:</code> init-keyword specifies the button on the attached pointer device on which the gesture has occurred, and the <code>modifier-state:</code> init-keyword specifies the modifier-state of the gesture.</p>
Operations	<code>gesture-button</code>

**<pointer-motion-event>***Sealed instantiable class*

Summary	The class of events that describe a pointer that is moving.
---------	---

Superclasses	<code>&lt;pointer-event&gt;</code>
Init-keywords	None.
Description	The class of events that describe a pointer that is moving.
Operations	None.
See also	<ul style="list-style-type: none"> <li><code>&lt;pointer-button-event&gt;</code>, page 280</li> <li><code>&lt;pointer-drag-event&gt;</code>, page 283</li> <li><code>&lt;pointer-enter-event&gt;</code>, page 283</li> <li><code>&lt;pointer-event&gt;</code>, page 284</li> <li><code>&lt;pointer-exit-event&gt;</code>, page 284</li> </ul>

		<i>Generic function</i>
Summary	Returns the current position of the specified pointer.	
Signature		<code>pointer-position <i>pointer</i> #key <i>sheet</i> =&gt; <i>x y</i></code>
Arguments	<ul style="list-style-type: none"> <li><i>pointer</i> An instance of type <code>&lt;pointer&gt;</code>.</li> <li><i>sheet</i> An instance of type <code>&lt;sheet&gt;</code>.</li> </ul>	
Values	<ul style="list-style-type: none"> <li><i>x</i> An instance of type <code>&lt;real&gt;</code>.</li> <li><i>y</i> An instance of type <code>&lt;real&gt;</code>.</li> </ul>	
Description	Returns the current position of <i>pointer</i> . If <i>sheet</i> is specified, then the pointer must be over it.	
See also	<ul style="list-style-type: none"> <li><code>pointer-sheet</code>, page 287</li> <li><code>set-pointer-position</code>, page 296</li> </ul>	

## **pointer-sheet** *Generic function*

Summary	Returns the sheet under the specified pointer.
Signature	<code>pointer-sheet <i>pointer</i> =&gt; <i>sheet</i></code>
Arguments	<i>pointer</i> An instance of type <code>&lt;pointer&gt;</code> .
Values	<i>sheet</i> An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Returns the sheet under <i>pointer</i> , or #f if there is no sheet under the pointer.
See also	<a href="#">pointer-position</a> , page 286

## **<port>** *Open abstract class*

Summary	The class of all ports.
Superclasses	<code>&lt;object&gt;</code>
Init-keywords	None.
Description	The class of all ports. A display, and all the sheets attached to a display, is associated with a port that is a connection to a display server. The port manages: <ul style="list-style-type: none"> <li>• A primary input device (usually a keyboard)</li> <li>• A pointing device, such as a mouse or trackball</li> <li>• An event processor that dispatched events to the appropriate sheet.</li> </ul>
Operations	The following operations are exported from the <code>DUIL-Sheets</code> module.

```
beep default-port-setter destroy-port force-display
get-default-background get-default-foreground get-
default-text-style port port? port-modifier-state
port-pointer port-server-path synchronize-display
text-size text-style-mapping text-style-mapping-
setter
```

The following operation is exported from the **DUIM-DCS** module.

**find-color**

See also      **<display>**, page 215  
                 **<sheet>**, page 299

<b>port</b>	<i>Generic function</i>
Summary	Returns the port for the specified object.
Signature	<b>port <i>object</i> =&gt; <i>value</i></b>
Arguments	<b><i>object</i></b> An instance of type <b>&lt;object&gt;</b> .
Values	<b><i>value</i></b> An instance of type <b>false-or(&lt;port&gt;)</b> .
Description	Returns the port used to display <i>object</i> .
See also	<b>display</b> , page 216 <b>frame-manager</b> , page 244 <b>&lt;port&gt;</b> , page 287 <b>port?</b> , page 289

## **port?** *Generic function*

Summary	Returns true if the specified object is a port.	
Signature	<code>port? object =&gt; boolean</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>object</i> is a port.	
See also	<a href="#">&lt;port&gt;, page 287</a> <a href="#">&lt;port&gt;, page 287</a>	

## **port-modifier-state** *Generic function*

Summary	Returns the modifier state of the specified port.	
Signature	<code>port-modifier-state port =&gt; integer</code>	
Arguments	<code>port</code>	An instance of type <code>&lt;port&gt;</code> .
Values	<code>integer</code>	An instance of type <code>&lt;integer&gt;</code> .
Description	Returns the modifier state of <i>port</i> .	
See also	<a href="#">event-modifier-state, page 234</a> <a href="#">gesture-modifier-state, page 248</a> <a href="#">make-modifier-state, page 258</a> <a href="#">port-name, page 290</a> <a href="#">port-pointer, page 290</a> <a href="#">port-server-path, page 291</a>	

**port-type**, page 292

**port-name** *Generic function*

Summary	Returns the name of the specified port.	
Signature	<b>port-name</b> <i>port</i> => <i>name</i>	
Arguments	<i>port</i>	An instance of type < <b>port</b> >.
Values	<i>name</i>	An instance of type < <b>object</b> >.
Description	Returns the name of <i>port</i> .	
See also	<b>port-modifier-state</b> , page 289 <b>port-pointer</b> , page 290 <b>port-server-path</b> , page 291 <b>port-type</b> , page 292	

**port-pointer** *Generic function*

Summary	Returns the pointer used on the specified port.	
Signature	<b>port-pointer</b> <i>port</i> => <i>pointer</i>	
Arguments	<i>port</i>	An instance of type < <b>port</b> >.
Values	<i>pointer</i>	An instance of type < <b>pointer</b> >.
Description	Returns the pointer used on <i>port</i> .	
See also	<b>port-modifier-state</b> , page 289 <b>port-name</b> , page 290	

**port-server-path**, page 291

**port-type**, page 292

## **port-server-path** *Generic function*

Summary      Returns the server path of the specified port.

Signature      **port-server-path** *port* => *object*

Arguments      *port*                  An instance of type <**port**>.

Values            *object*                  An instance of type <**object**>.

Description      Returns the server path of *port*.

See also        **port-modifier-state**, page 289

**port-name**, page 290

**port-pointer**, page 290

**port-type**, page 292

## **<port-terminated-event>** *Sealed instantiable class*

Summary      The class of events that describe the termination of a port.

Superclasses    <**frame-exited-event**>

Init-keywords    **condition:**      An instance of type <**condition**>. Required.

Description      The class of events that describe the termination of a port.

The **condition:** init-keyword returns the error condition signalled when the port was terminated.

Operations      None.

**port-type** *Generic function*

Summary	Returns the type of the specified port.	
Signature	<code>port-type <i>port</i> =&gt; <i>type</i></code>	
Arguments	<i>port</i>	An instance of type < <code>port</code> >.
Values	<i>type</i>	An instance of type < <code>symbol</code> >.
Description	Returns the type of <i>port</i> .	
See also	<a href="#">port-modifier-state</a> , page 289 <a href="#">port-name</a> , page 290 <a href="#">port-pointer</a> , page 290 <a href="#">port-server-path</a> , page 291	

**queue-event** *Generic function*

Summary	Queues an event for the specified sheet.	
Signature	<code>queue-event <i>sheet</i> <i>event</i> =&gt; ()</code>	
Arguments	<i>sheet</i>	An instance of type < <code>sheet</code> >.
	<i>event</i>	An instance of type < <code>event</code> >.
Values	None	
Description	Queues <i>event</i> on the event-queue for <i>sheet</i> .	
See also	<a href="#">handle-event</a> , page 252	

## queue-repaint *Generic function*

Summary	Queues a repaint for the specified region of the specified sheet.	
Signature	<code>queue-repaint <i>sheet region</i> =&gt; ()</code>	
Arguments	<i>sheet</i>	An instance of type < <code>sheet</code> >.
	<i>region</i>	An instance of type < <code>region</code> >.
Values	None	
Description	Queues a repaint for the area of <i>sheet</i> defined by <i>region</i> .	
See also	<a href="#">handle-repaint</a> , page 253 <a href="#">repaint-sheet</a> , page 294 <a href="#">&lt;window-repaint-event&gt;</a> , page 326	

## raise-sheet *Generic function*

Summary	Raises the specified sheet to the top of the current hierarchy of sheets.	
Signature	<code>raise-sheet <i>sheet</i> =&gt; ()</code>	
Arguments	<i>sheet</i>	An instance of type < <code>sheet</code> >.
Values	None	
Description	Raises <i>sheet</i> to the top of the current hierarchy of sheets.	
See also	<a href="#">lower-frame</a> , page 727 <a href="#">lower-sheet</a> , page 257 <a href="#">raise-frame</a> , page 741	

**remove-child** *Generic function*

Summary	Removes a child from the specified sheet.	
Signature	<code>remove-child <i>sheet</i> <i>child</i> =&gt; <i>sheet</i></code>	
Arguments	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
	<i>child</i>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<i>sheet</i> An instance of type <code>&lt;sheet&gt;</code> .	
Description	Removes <i>child</i> from <i>sheet</i> . The remaining children in the sheet are laid out again appropriately.	
See also	<a href="#">add-child</a> , page 184 <a href="#">replace-child</a> , page 295	

**repaint-sheet** *Generic function*

Summary	Repaints the specified region of a sheet.	
Signature	<code>repaint-sheet <i>sheet</i> <i>region</i> #key <i>medium</i> =&gt; ()</code>	
Arguments	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
	<i>region</i>	An instance of type <code>&lt;region&gt;</code> .
	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .
Values	None	
Description	Repaints the area of <i>sheet</i> defined by <i>region</i> . If specified, the appropriate <i>medium</i> is used.	
See also	<a href="#">handle-repaint</a> , page 253 <a href="#">queue-repaint</a> , page 293	

`<window-repaint-event>`, page 326

## replace-child *Generic function*

Summary	Replaces a child from the specified sheet with a new one.	
Signature	<code>replace-child sheet old-child new-child =&gt; sheet</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>old-child</code>	An instance of type <code>&lt;object&gt;</code> .
	<code>new-child</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
Description	Replaces <i>old-child</i> with <i>new-child</i> in <i>sheet</i> . The children in the sheet are laid out again appropriately.	
See also	<a href="#">add-child</a> , page 184 <a href="#">remove-child</a> , page 294	

## \$right-button *Constant*

Summary	A constant that represents the right button on the attached pointing device.	
Type	<code>&lt;integer&gt;</code>	
Value	<code>ash(1, %button_base + 2)</code>	
Description	A constant that represents the right button on the attached pointing device.	
See also	<a href="#">\$left-button</a> , page 257	

`$middle-button`, page 272

`$pointer-buttons`, page 281

## set-caret-position

*Generic function*

Summary      Sets the position of the specified cursor.

Signature      `set-cursor-position cursor x y => ()`

Arguments      `cursor`      An instance of type `<caret>`.

`x`      An instance of type `<real>`.

`y`      An instance of type `<real>`.

Values      None

Description      Sets the position of `cursor` to  $(x, y)$ .

See also      [caret-position](#), page 190

[set-pointer-position](#), page 296

## set-pointer-position

*Generic function*

Summary      Sets the position of the specified pointer.

Signature      `set-pointer-position pointer x y #key sheet => ()`

Arguments      `pointer`      An instance of type `<pointer>`.

`x`      An instance of type `<real>`.

`y`      An instance of type `<real>`.

`sheet`      An instance of type `<sheet>`.

Values      None

Description Sets the position of *pointer* to  $(x, y)$ , relative to the top left corner of *sheet*, if specified. Units are measured in pixels.

See also [pointer-position, page 286](#)  
[set-pointer-position, page 296](#)

## set-sheet-edges *Generic function*

Summary Sets the edges of the specified sheet relative to its parent.

Signature **set-sheet-edges** *sheet left top right bottom* => ()

Arguments

<i>sheet</i>	An instance of type < <i>sheet</i> >.
<i>left</i>	An instance of type < <i>integer</i> >.
<i>top</i>	An instance of type < <i>integer</i> >.
<i>right</i>	An instance of type < <i>integer</i> >.
<i>bottom</i>	An instance of type < <i>integer</i> >.

Values None

Description Sets the edges of *sheet* to *top*, *left*, *right*, and *bottom*. Each edge is specified relative to the corresponding edge of the parent of *sheet*. The layout of *sheet* is recalculated automatically.

See also [set-sheet-position, page 297](#)  
[set-sheet-size, page 298](#)  
[sheet-edges, page 306](#)

## set-sheet-position *Generic function*

Summary Sets the position of the specified sheet relative to its parent.

Signature	<code>set-sheet-position sheet x y =&gt; ()</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>x</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>y</code>	An instance of type <code>&lt;real&gt;</code> .
Values	None	
Description	Sets the position of <code>sheet</code> to $(x, y)$ relative to the position of its parent. The layout of <code>sheet</code> is recalculated automatically.	
See also	<a href="#">set-sheet-edges, page 297</a> <a href="#">set-sheet-size, page 298</a> <a href="#">sheet-position, page 313</a>	

## **set-sheet-size**

*Generic function*

Summary	Sets the size of the specified sheet.						
Signature	<code>set-sheet-size sheet width height =&gt; ()</code>						
Arguments	<table> <tr> <td><code>sheet</code></td> <td>An instance of type <code>&lt;sheet&gt;</code>.</td> </tr> <tr> <td><code>width</code></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> <tr> <td><code>height</code></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> </table>	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .	<code>width</code>	An instance of type <code>&lt;integer&gt;</code> .	<code>height</code>	An instance of type <code>&lt;integer&gt;</code> .
<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .						
<code>width</code>	An instance of type <code>&lt;integer&gt;</code> .						
<code>height</code>	An instance of type <code>&lt;integer&gt;</code> .						
Values	None						
Description	Sets the size of <code>sheet</code> . The layout of <code>sheet</code> is recalculated automatically.						
See also	<a href="#">set-sheet-edges, page 297</a> <a href="#">set-sheet-position, page 297</a>						

**<sheet>***Open abstract class*

Summary	The base object class for DUIM windows.
Superclasses	<b>&lt;object&gt;</b>
Init-keywords	<p><b>region:</b> An instance of type <b>&lt;region&gt;</b>. Default value <b>\$nowhere</b>.</p> <p><b>transform:</b> An instance of type <b>&lt;transform&gt;</b>. Default value <b>\$identity-transform</b>.</p> <p><b>port:</b> An instance of type <b>false-or(&lt;port&gt;)</b>. Default value <b>#f</b>.</p> <p><b>style-descriptor:</b> An instance of type <b>false-or(style-descriptor)</b>. Default value <b>#f</b>.</p> <p><b>help-context:</b> An instance of type <b>&lt;object-table&gt;</b>. Default value <b>make(&lt;object-table&gt;)</b>.</p> <p><b>help-source:</b> An instance of type <b>&lt;object-table&gt;</b>. Default value <b>make(&lt;object-table&gt;)</b>.</p> <p><b>parent:</b> An instance of type <b>false-or(&lt;sheet&gt;)</b>. Default value: <b>#f</b>.</p> <p><b>child:</b> An instance of type <b>false-or(&lt;sheet&gt;)</b>. Default value: <b>#f</b>.</p> <p><b>children:</b> An instance of type <b>limited(&lt;sequence&gt;, of: &lt;sheet&gt;)</b>. Default value: <b>#[]</b>.</p> <p><b>x:</b> An instance of type <b>&lt;integer&gt;</b>.</p> <p><b>y:</b> An instance of type <b>&lt;integer&gt;</b>.</p> <p><b>withdrawn?:</b> An instance of type <b>&lt;boolean&gt;</b>. Default value: <b>#f</b>.</p>

	<b>accepts-focus?:</b>	An instance of type <boolean>. Default value: #t.
	<b>cursor:</b>	An instance of type <cursor>.
	<b>caret:</b>	An instance of type type-union(<caret>, one-of(#f, #t)). Default value: #f.
	<b>foreground:</b>	An instance of type <ink>.
	<b>background:</b>	An instance of type <ink>.
	<b>text-style:</b>	An instance of type <text-style>.
	<b>fixed-width?:</b>	An instance of type <boolean>.
	<b>fixed-height?:</b>	An instance of type <boolean>.
	<b>resizable?:</b>	An instance of type <boolean>.
Description	<p>The <b>port</b>: init-keyword is true if the pane (and its mirror, if it has one) has been mapped, #f otherwise. In this case, the term <i>mapped</i> means visible on the display, ignoring issues of occlusion.</p> <p>The <b>help-source</b>: and <b>help-context</b>: keywords let you specify pointers to valid information available in any online help you supply with your application. The <b>help-context</b>: keyword should specify a context-ID present in the online help. This context-ID identifies the help topic that is applicable to the current pane. The <b>help-source</b>: init-keyword identifies the source file in which the help topic identified by <b>help-context</b>: can be found. A list of context-IDs should be provided by the author of the online help system.</p> <p>The <b>parent</b>:, <b>child</b>:, and <b>children</b>: init-keywords let you specify a lineage for the sheet if you wish, specifying the parent of the sheet and as many children as you wish.</p> <p>The <b>x</b>: and <b>y</b>: init-keywords specify the initial position of the sheet relative to its parent. When <b>accepts-focus</b>: is true, the sheet will accept the pointer focus.</p>	

The init-keywords `cursor:`, `foreground:`, `background:`, and `text-style:` can be used to specify the appearance of elements in the sheet.

The `caret:` init-keyword is used to specify the caret to be used within the drawing pane, if one is to be used at all.

The `fixed-width?:` and `fixed-height?:` init-keywords are used to fix the width or height of a sheet to the size defined by other appropriate init-keywords. This is a useful way of ensuring that the default size defined for a sheet is fixed in either direction. The init-keywords force the space requirements for the sheet to make the minimum and maximum sizes equal to the size defined at the time of creation. These keywords are most useful when creating sheets of unknown size, when you want to ensure that any child of that sheet is fixed at that size, whatever it may be.

If `resizable?:` is `#t` then the sheet can be resized in either direction. If `resizable?:` is `#f` then it cannot be resized in either direction. If `resizable?:` is `#t`, but one of `fixed-width?:` or `fixed-height?:` is `#t`, then the sheet can only be resized in one direction as appropriate.

## Operations

The following operations are exported from the `DUIM-Sheets` module.

```
add-child beep child-containing-position children-overlapping-region clear-box curve-to destroy-sheet display do-children-containing-position do-children-overlapping-region do-sheet-children do-sheet-tree do-with-drawing-options do-with-pointer-grabbed do-with-sheet-medium do-with-text-style do-with-transform force-display frame-manager get-default-background get-default-foreground get-default-text-style handle-event handle-repaint
```

```
medium-background medium-background-setter medium-
brush medium-brush-setter medium-clipping-region
medium-clipping-region-setter medium-default-text-
style medium-default-text-style-setter medium-fore-
ground medium-foreground-setter medium-pen medium-
pen-setter medium-text-style
medium-text-style-setter medium-transform medium-
transform-setter port

queue-event queue-repaint

raise-sheet remove-child repaint-sheet replace-child

set-sheet-edges set-sheet-position set-sheet-size
sheet?

sheet-ancestor? sheet-child sheet-children sheet-
children-setter sheet-child-setter sheet-edges sheet-
frame sheet-mapped? sheet-mapped?-setter sheet-medium
sheet-parent sheet-parent-setter sheet-position
sheet-region sheet-region-setter sheet-size sheet-
state sheet-transform sheet-transform-setter sheet-
viewport sheet-viewport-region sheet-withdrawn?

synchronize-display text-size top-level-sheet
```

The following operations are exported from the **DUIM-Gadgets** module.

```
scroll-position set-scroll-position
```

The following operations are exported from the **DUIM-Layouts** module.

```
allocate-space compose-space

do-allocate-space do-compose-space

relayout-children relayout-parent

space-requirement-height space-requirement-max-
height space-requirement-max-width space-requirement-
min-height space-requirement-min-width
space-requirement-width
```

The following operations are exported from the **DUIM-Frames** module.

**exit-dialog**

The following operations are exported from the **DUIM-Graphics** module.

```
abort-path arc-to close-path copy-area curve-to do-
with-output-to-pixmap draw-bezier-curve draw-ellipse
draw-image draw-line draw-lines draw-pixmap draw-point
draw-points draw-polygon draw-rectangle draw-text end-
path fill-path line-to move-to restore-clipping-region
start-path stroke-path
```

The following operations are exported from the **DUIM-DCS** module.

```
default-background default-foreground default-text-
style
```

The following operations are exported from the **DUIM-Geometry** module.

**box-edges**

The following operations are exported from the **DUIM-Extended-Geometry** module.

**draw-design**

Examples To make a text editor that is fixed at 10 lines high:

```
make(<text-editor>, lines: 10, fixed-height?: #t);
```

See also **<display>**, page 215

**<port>**, page 287

**sheet?**

*Generic function*

Summary Returns true if the specified object is a sheet.

Signature	<code>sheet? object =&gt; boolean</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <code>object</code> is a sheet.	
See also	<code>medium?</code> , page 261	

## **sheet-ancestor?** *Generic function*

Summary	Returns true if the specified sheet has the specified ancestor.	
Signature	<code>sheet-ancestor? sheet putative-ancestor =&gt; boolean</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>putative-ancestor</code>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <code>putative-ancestor</code> is an ancestor of <code>sheet</code> .	
See also	<code>sheet?</code> , page 303	

## **sheet-child** *Generic function*

Summary	Returns the child of the specified sheet.	
Signature	<code>sheet-child sheet =&gt; child</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .

Values	<i>child</i>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Returns the child of <i>sheet</i> .	
See also	<a href="#">sheet-children, page 305</a> <a href="#">sheet-child-setter, page 306</a>	

## **sheet-children** *Generic function*

Summary	Returns a list of sheets that are the children of the specified sheet.	
Signature	<code>sheet-children sheet =&gt; sheets</code>	
Arguments	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<i>sheets</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;sheet&gt;)</code> .
Description	Returns a list of sheets that are the children of <i>sheet</i> . Some sheet classes support only a single child; in this case, the return value of <code>sheet-children</code> is a list of one element.	
See also	<a href="#">do-sheet-children, page 225</a> <a href="#">sheet-child, page 304</a> <a href="#">sheet-children-setter, page 305</a>	

## **sheet-children-setter** *Generic function*

Summary	Sets the children of the specified sheet.	
Signature	<code>sheet-children-setter children sheet =&gt; sheets</code>	

Arguments	<i>children</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;sheet&gt;)</code> .
	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<i>children</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;sheet&gt;)</code> .
Description		Sets the children of <i>sheet</i> . Some sheet classes support only a single child; in this case, <i>children</i> is a list of one element.
See also		<code>sheets-children</code> , page 305 <code>sheets-child-setter</code> , page 306

## **sheets-child-setter** *Generic function*

Summary	Sets the child of the specified sheet.
Signature	<code>sheets-child-setter child sheet =&gt; child</code>
Arguments	<i>child</i> An instance of type <code>&lt;sheet&gt;</code> . <i>sheet</i> An instance of type <code>&lt;sheet&gt;</code> .
Values	<i>child</i> An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Sets the child of <i>sheet</i> .
See also	<code>sheets-child</code> , page 304 <code>sheets-children-setter</code> , page 305

## **sheets-edges** *Generic function*

Summary	Returns the edges of the specified sheet, relative to its parent.
---------	---

Signature	<code>sheet-edges sheet =&gt; left top right bottom</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>left</code>	An instance of type <code>&lt;coordinate&gt;</code> .
	<code>top</code>	An instance of type <code>&lt;coordinate&gt;</code> .
	<code>right</code>	An instance of type <code>&lt;coordinate&gt;</code> .
	<code>bottom</code>	An instance of type <code>&lt;coordinate&gt;</code> .
Description	Returns the edges of <code>sheet</code> . Each edge is specified relative to the corresponding edge of the parent of <code>sheet</code> .	
See also	<a href="#">set-sheet-edges</a> , page 297 <a href="#">sheet-position</a> , page 313 <a href="#">sheet-size</a> , page 314 <a href="#">sheet-transform</a> , page 316	

## `<sheet-event>`

*Open abstract class*

Summary	The class of events that can occur in sheets.
Superclasses	<code>&lt;event&gt;</code>
Init-keywords	<code>sheet:</code> An instance of type <code>false-or(&lt;sheet&gt;)</code> . Required.
Description	<p>The class of events that can occur in sheets.</p> <p>The required init-keyword <code>sheet:</code> specifies a sheet in which the event occurs.</p>
Operations	<p>The following operation is exported from the <code>DUIM-Sheets</code> module.</p> <p><code>event-sheet</code></p>

See also <**device-event**>, page 214

## sheet-event-mask

## *Generic function*

**Summary** Returns the event mask of the specified sheet.

Signature      **sheet-event-mask** *sheet* => *integer*

Arguments      *sheet*                  An instance of type `<sheet>`.

Values      *integer*      An instance of type `<integer>`.

Description Returns the event mask of *sheet*.

See also [sheet-event-mask-setter](#), page 308

**sheet-event-queue**, page 309

## **sheet-event-mask-setter**

### *Generic function*

**Summary** Sets the event mask of the specified sheet.

**Signature**      `sheet-event-mask-setter mask sheet => mask`

Arguments      *mask*      An instance of type <integer>

*sheet* An instance of type <sheet>.

*sheet* An instance of type <sheet>.

**Values**      *mask*      An instance of type <integer>.

Description Sets the event mask of *sheet*.

See also [sheet-event-mask](#), page 308

**sheet-event-queue** *Generic function*

Summary	Returns the event queue of the specified sheet.
Signature	<code>sheet-event-queue sheet =&gt; event-queue</code>
Arguments	<code>sheet</code> An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>event-queue</code> An instance of type <code>&lt;event-queue&gt;</code> .
Description	Returns the event mask of <code>sheet</code> . This is a list of all the events that are currently queued ready for execution.
See also	<code>sheet-event-mask</code> , page 308

**sheet-frame** *Generic function*

Summary	Returns the frame associated with the specified sheet.
Signature	<code>sheet-frame sheet =&gt; frame</code>
Arguments	<code>sheet</code> An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>frame</code> An instance of type <code>false-or(&lt;frame&gt;)</code> .
Description	Returns the frame associated with <code>sheet</code> .
See also	<code>sheet-medium</code> , page 310 <code>sheet-parent</code> , page 311

**sheet-mapped?** *Generic function*

Summary	Returns true if the specified sheet is mapped.
---------	--

Signature	<code>sheet-mapped? sheet =&gt; mapped?</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>mapped?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <code>sheet</code> is mapped, that is, displayed on screen (issues of occluding windows notwithstanding).	
See also	<code>sheet-mapped?-setter</code> , page 310 <code>sheet-withdrawn?</code> , page 317	

## **sheet-mapped?-setter** *Generic function*

Summary	Specifies whether the specified sheet is mapped.	
Signature	<code>sheet-mapped?-setter mapped? sheet =&gt; boolean</code>	
Arguments	<code>mapped?</code>	An instance of type <code>&lt;boolean&gt;</code> .
	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>boolean</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Specifies whether <code>sheet</code> is mapped, that is, displayed on screen (issues of occluding windows notwithstanding). If #t, <code>sheet</code> is mapped, if #f, it is not.	
See also	<code>sheet-mapped?</code> , page 309	

## **sheet-medium** *Generic function*

Summary	Returns the medium associated with the specified sheet.	
Signature	<code>sheet-medium sheet =&gt; medium</code>	

Arguments	<i>sheet</i>	An instance of type < <i>sheet</i> >.
Values	<i>medium</i>	An instance of type <code>false-or(&lt;medium&gt;)</code> .
Description		Returns the medium associated with <i>sheet</i> .
See also		<code>sheet-frame</code> , page 309

## **sheet-parent** *Generic function*

Summary	Returns the parent of the specified sheet.	
Signature	<code>sheet-parent <i>sheet</i> =&gt; <i>parent</i></code>	
Arguments	<i>sheet</i>	An instance of type < <i>sheet</i> >.
Values	<i>parent</i>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description		Returns the parent of <i>sheet</i> .
See also	<a href="#">sheet-medium</a> , page 310 <a href="#">sheet-parent-setter</a> , page 311 <a href="#">sheet-position</a> , page 313	

## **sheet-parent-setter** *Generic function*

Summary	Sets the parent of the specified sheet.	
Signature	<code>sheet-parent-setter <i>parent sheet</i> =&gt; <i>value</i></code>	
Arguments	<i>parent</i>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
	<i>sheet</i>	An instance of type < <i>sheet</i> >.

Values	<i>value</i>	An instance of type <b>false-or(&lt;sheet&gt;)</b> .
Description	Sets the parent of <i>sheet</i> .	
See also		<a href="#">sheet-parent</a> , page 311

## **sheet-pointer-cursor** *Generic function*

Summary	Returns the pointer cursor associated with the specified sheet.	
Signature	<b>sheet-pointer-cursor</b> <i>sheet</i> => <i>cursor</i>	
Arguments	<i>sheet</i>	An instance of type <b>&lt;sheet&gt;</b> .
Values	<i>cursor</i>	An instance of type <b>&lt;cursor&gt;</b> .
Description	Returns the pointer cursor associated with <i>sheet</i> . This is the cursor used to represent the mouse pointer whenever the mouse pointer is inside the boundary of <i>sheet</i> .	
See also	<a href="#">sheet-pointer-cursor-setter</a> , page 312 <a href="#">sheet-text-cursor</a> , page 315	

## **sheet-pointer-cursor-setter** *Generic function*

Summary	Sets the pointer cursor associated with the specified sheet.	
Signature	<b>sheet-pointer-cursor-setter</b> <i>cursor sheet</i> => <i>cursor</i>	
Arguments	<i>cursor</i>	An instance of type <b>&lt;cursor&gt;</b> .
	<i>sheet</i>	An instance of type <b>&lt;sheet&gt;</b> .
Values	<i>cursor</i>	An instance of type <b>&lt;cursor&gt;</b> .

Description Sets the pointer cursor associated with *sheet*. This is the cursor used to represent the mouse pointer whenever the mouse pointer is inside the boundary of *sheet*.

See also [sheet-pointer-cursor](#), page 312

## **sheet-position** *Generic function*

Summary Returns the position of the specified sheet relative to its parent.

Signature `sheet-position sheet => xy`

Arguments `sheet` An instance of type `<sheet>`.

Values `x` An instance of type `<real>`.

`y` An instance of type `<real>`.

Description Returns the position of *sheet*. The position is represented by the coordinate (x,y), as measured relative to the parent of *sheet*, or relative to the top left of the screen if *sheet* has no parent.

See also [set-sheet-position](#), page 297

[sheet-edges](#), page 306

[sheet-parent](#), page 311

[sheet-size](#), page 314

[sheet-transform](#), page 316

## **sheet-region** *Generic function*

Summary Returns the region associated with the specified sheet.

Signature	<b>sheet-region</b> <i>sheet</i> => <i>region</i>	
Arguments	<i>sheet</i>	An instance of type < <i>sheet</i> >.
Values	<i>region</i>	An instance of type < <i>region</i> >.
Description	Returns an instance of < <i>region</i> > that represents the set of points to which <i>sheet</i> refers. The region is expressed in the same coordinate system as <i>sheet</i> .	
See also	<a href="#">sheet-region-setter</a> , page 314	

## **sheet-region-setter** *Generic function*

Summary	Sets the region associated with the specified sheet.	
Signature	<b>sheet-region-setter</b> <i>region sheet</i> => <i>region</i>	
Arguments	<i>region</i>	An instance of type < <i>region</i> >.
	<i>sheet</i>	An instance of type < <i>sheet</i> >.
Values	<i>region</i>	An instance of type < <i>region</i> >.
Description	Creates or modifies an instance of < <i>region</i> > that represents the set of points to which <i>sheet</i> refers. The region is expressed in the same coordinate system as <i>sheet</i> .	
See also	<a href="#">sheet-region</a> , page 313	

## **sheet-size** *Generic function*

Summary	Returns the width and height of the specified sheet.	
Signature	<b>sheet-size</b> <i>sheet</i> => <i>width height</i>	

Arguments	<i>sheet</i>	An instance of type <sheet>.
Values	<i>width</i>	An instance of type <integer>.
	<i>height</i>	An instance of type <integer>.
Description	Returns the width and height of the specified sheet. Use <code>set-sheet-size</code> to set or modify the size of a sheet.	
See also	<a href="#">set-sheet-size</a> , page 298 <a href="#">sheet-edges</a> , page 306 <a href="#">sheet-position</a> , page 313 <a href="#">sheet-transform</a> , page 316	

## **sheet-state** *Generic function*

Summary	Returns the current state of the specified sheet.
Signature	<code>sheet-state sheet =&gt; value</code>
Arguments	<i>sheet</i>
Values	<i>value</i>
	An instance of type one-of(#"withdrawn", #"managed", #"mapped", #"unknown").
Description	Returns the current state of <i>sheet</i> . The state of a sheet tells you whether the sheet is currently mapped on screen, or whether it has been withdrawn from the list of sheets.

## **sheet-text-cursor** *Generic function*

Summary	Returns the text cursor associated with the specified sheet.
Signature	<code>sheet-text-cursor sheet =&gt; text-cursor</code>

Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>text-cursor</code>	An instance of type <code>false-or(&lt;cursor&gt;)</code> .
Description		Returns the text cursor associated with <code>sheet</code> . The text cursor associated with a sheet is distinct from the pointer cursor associated with the same sheet: the pointer cursor represents the current position of the pointer associated with the attached pointer device, while the text cursor represents the position in the sheet that any text typed using the keyboard will be added. Only those sheets that contain children that allow some form of text-based input have an associated text cursor.
See also		<code>sheet-pointer-cursor</code> , page 312

		<i>Generic function</i>
Summary		Returns the transform associated with the specified sheet.
Signature		<code>sheet-transform sheet =&gt; transform</code>
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>transform</code>	An instance of type <code>&lt;transform&gt;</code> .
Description		Returns the transform associated with <code>sheet</code> .
See also		<code>medium-transform</code> , page 271 <code>sheet-edges</code> , page 306 <code>sheet-position</code> , page 313 <code>sheet-size</code> , page 314

## **sheet-transform-setter** *Generic function*

Summary	Sets the transform associated with the specified sheet.	
Signature	<code>sheet-transform-setter</code> <i>transform sheet =&gt; transform</i>	
Arguments	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<i>transform</i>	An instance of type <code>&lt;transform&gt;</code> .
Description	Sets or modifies the transform associated with <i>sheet</i> .	
See also	<a href="#">medium-transform-setter</a> , page 271	

## **sheet-withdrawn?** *Generic function*

Summary	Returns true if the specified sheet has been withdrawn from the display.	
Signature	<code>sheet-withdrawn?</code> <i>sheet =&gt; withdrawn?</i>	
Arguments	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<i>withdrawn?</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>sheet</i> has been withdrawn from the display, and is no longer mapped.	
See also	<a href="#">sheet-mapped?</a> , page 309	

## **\$shift-key** *Constant*

Summary	A constant that represents the SHIFT key on the keyboard.
---------	---

Type	<code>&lt;integer&gt;</code>
Value	<code>ash(1, %modifier_base + 0);</code>
Description	A constant that represents the SHIFT key on the keyboard.
See also	<ul style="list-style-type: none"> <li><a href="#"><code>\$alt-key</code>, page 185</a></li> <li><a href="#"><code>\$control-key</code>, page 211</a></li> <li><a href="#"><code>\$hyper-key</code>, page 254</a></li> <li><a href="#"><code>\$meta-key</code>, page 271</a></li> <li><a href="#"><code>modifier-key-index</code>, page 273</a></li> <li><a href="#"><code>modifier-key-index-name</code>, page 273</a></li> <li><a href="#"><code>\$modifier-keys</code>, page 274</a></li> <li><a href="#"><code>\$option-key</code>, page 278</a></li> <li><a href="#"><code>\$super-key</code>, page 318</a></li> </ul>

	<i>Constant</i>
<b>\$super-key</b>	
Summary	A constant that represents the SUPER key on the keyboard.
Type	<code>&lt;integer&gt;</code>
Value	<code>ash(1, %modifier_base + 3);</code>
Description	A constant that represents the SUPER key on the keyboard, if it exists. To deal with the case where there is no SUPER key, the value of the constant <code>\$option-key</code> is bound to this constant.
See also	<ul style="list-style-type: none"> <li><a href="#"><code>\$alt-key</code>, page 185</a></li> <li><a href="#"><code>\$control-key</code>, page 211</a></li> <li><a href="#"><code>\$hyper-key</code>, page 254</a></li> </ul>

**\$meta-key**, page 271  
**modifier-key-index**, page 273  
**modifier-key-index-name**, page 273  
**\$modifier-keys**, page 274  
**\$option-key**, page 278  
**\$shift-key**, page 317

## **synchronize-display** *Generic function*

Summary	Synchronizes all displays on which the specified drawable is mapped.
Signature	<b>synchronize-display</b> <i>drawable</i> => ()
Arguments	<i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
Values	None
Description	Synchronizes all displays on which the specified drawable is mapped.

## **text-size** *Generic function*

Summary	Returns information about the size of the specified text on the specified medium.
Signature	<b>text-size</b> <i>medium</i> <i>text</i> #key <i>text-style</i> <i>start</i> <i>end</i> <i>do-newlines?</i> => <i>largest-x</i> <i>largest-y</i> <i>cursor-x</i> <i>cursor-y</i> <i>baseline</i>
Arguments	<i>medium</i> An instance of type <code>&lt;medium&gt;</code> . <i>text</i> An instance of type <code>type-union(&lt;string&gt;, &lt;character&gt;)</code> .

	<i>text-style</i>	An instance of type < <code>text-style</code> >.
	<i>start</i>	An instance of type < <code>integer</code> >. Default value: 0.
	<i>end</i>	An instance of type < <code>integer</code> >. Default value: <code>size(text)</code> .
	<i>do-newlines?</i>	An instance of type < <code>boolean</code> >. Default value: #f.
	<i>do-tabs?</i>	An instance of type < <code>boolean</code> >. Default value: #f.
Values	<i>largest-x</i>	An instance of type < <code>integer</code> >.
	<i>total-height</i>	An instance of type < <code>integer</code> >.
	<i>last-x</i>	An instance of type < <code>integer</code> >.
	<i>last-y</i>	An instance of type < <code>integer</code> >.
	<i>baseline</i>	An instance of type < <code>integer</code> >.
Description	<p>Returns information about the size of <i>text</i> on <i>medium</i>. If <i>text-style</i> is specified, then the information that <code>text-size</code> returns is based on the text style it describes.</p> <p>If <i>start</i> and <i>end</i> are specified, then these values represent a portion of the string specified by <i>text</i>, and only the characters they represent are examined by <code>text-size</code>. Both <i>start</i> and <i>end</i> represent the index of each character in <i>text</i>, starting at 0. By default, the whole of <i>text</i> is examined.</p> <p>The <i>do-newlines?</i> and <i>do-tabs?</i> arguments let you specify how newline or tab characters in <i>text</i> should be handled. If either of these arguments is true, then any newline or tab characters in <i>text</i> are examined, as appropriate. By default, newline characters are ignored.</p>	

## **text-style-mapping** *Generic function*

Summary	Returns the mapping for the specified text style on the specified port.	
Signature	<code>text-style-mapping <i>port</i> <i>text-style</i> #key <i>character-set</i> =&gt; <i>font</i></code>	
Arguments	<i>port</i>	An instance of type <code>&lt;port&gt;</code> .
	<i>text-style</i>	An instance of type <code>&lt;text-style&gt;</code> .
	<i>character-set</i>	An instance of type <code>&lt;object&gt;</code> .
Values	<i>font</i>	An instance of type <code>&lt;object&gt;</code> .
Description	<p>Returns the mapping for <i>text-style</i> on <i>port</i>. Mapping text styles onto fonts lets you control how different text styles are displayed on different servers, depending on the connection. For instance, it is possible to define how colored text is displayed on monochrome displays, or how fonts specified by <i>text-style</i> are mapped onto fonts available on the display.</p> <p>If <i>character-set</i> is specified, then this character set is used instead of the default. This is most useful for non-English displays.</p>	
See also	<a href="#">text-style-mapping-exists?</a> , page 321 <a href="#">text-style-mapping-setter</a> , page 322 <a href="#"><code>&lt;undefined-text-style-mapping&gt;</code></a> , page 324	

## **text-style-mapping-exists?** *Generic function*

Summary	Returns true if a mapping exists for the specified text style on the specified port.
Signature	<code>text-style-mapping-exists? <i>port</i> <i>text-style</i> #key <i>character-set</i> <i>exact-size?</i> =&gt; boolean</code>

Arguments	<i>port</i>	An instance of type < <code>port</code> >.
	<i>text-style</i>	An instance of type < <code>text-style</code> >.
	<i>character-set</i>	An instance of type < <code>object</code> >.
	<i>exact-size?</i>	An instance of type < <code>boolean</code> >. Default value: #f.
Values	<i>boolean</i>	An instance of type < <code>boolean</code> >.
Description	Returns true if a mapping exists for <i>text-style</i> on <i>port</i> . This control function is useful if, for example, you are setting up text style mappings for a range of text styles in one go, or for a range of different ports. Using this function, you can test for the existence of a previous mapping before creating a new one, thereby ensuring that existing mappings are not overwritten.	
See also	<a href="#">text-style-mapping</a> , page 321 <a href="#">text-style-mapping-setter</a> , page 322 <a href="#">&lt;undefined-text-style-mapping&gt;</a> , page 324	

## text-style-mapping-setter *Generic function*

Summary	Sets the mapping for the specified text style on the specified port.	
Signature	<code>text-style-mapping-setter font port text-style #key character-set =&gt; font</code>	
Arguments	<i>font</i>	An instance of type < <code>object</code> >.
	<i>port</i>	An instance of type < <code>port</code> >.
	<i>text-style</i>	An instance of type < <code>text-style</code> >.
	<i>character-set</i>	An instance of type < <code>object</code> >.

Values	<i>font</i>	An instance of type <object>.
Description	<p>Sets the mapping for <i>text-style</i> on <i>port</i> to the specified <i>font</i>. This function lets you have some control over the way in which different text styles are displayed on different servers, depending on the connection. Using this function, for instance, it is possible to define how colored text is displayed on monochrome displays, or how fonts specified by <i>text-style</i> are mapped onto fonts available on the display.</p> <p>If <i>character-set</i> is specified, then this character set is used instead of the default. This is most useful for non-English displays.</p>	
See also	<p><a href="#">text-style-mapping</a>, page 321</p> <p><a href="#">text-style-mapping-exists?</a>, page 321</p> <p><a href="#">&lt;undefined-text-style-mapping&gt;</a>, page 324</p>	

## <timer-event> *Sealed instantiable class*

Summary      The class of timed events.

Superclasses    <[frame-event](#)>

Init-keywords   None.

Description      The class of timed events.

Operations     None.

## top-level-sheet *Generic function*

Summary      Returns the top level sheet for the specified object.

Signature	<code>top-level-sheet object =&gt; top-level-sheet</code>	
Arguments	<code>object</code>	An instance of type <code>&lt;object&gt;</code> .
Values	<code>top-level-sheet</code>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Returns the top level sheet for <code>object</code> . This is the sheet that has as its descendants all of the panes of <code>object</code> .	

**<undefined-text-style-mapping>** *Sealed instantiable class*

Summary	The class of undefined text style mappings.
Superclasses	<code>&lt;error&gt;</code>
Init-keywords	<code>port:</code> An instance of type <code>&lt;port&gt;</code> . Required. <code>text-style:</code> An instance of type <code>&lt;text-style&gt;</code> . Required.
Description	The class of undefined text style mappings. This class is used for any text styles that have not had mappings defined for a given port.
Operations	None.
See also	<code>text-style-mapping</code> , page 321 <code>text-style-mapping-exists?</code> , page 321 <code>text-style-mapping-setter</code> , page 322

**<>window-configuration-event>** *Sealed instantiable class*

Summary	The class of events involving changes to the window configuration.
---------	--

Superclasses	<code>&lt;window-event&gt;</code>
Init-keywords	None.
Description	The class of events involving changes to the window configuration.
Operations	None.
See also	<code>&lt;window-repaint-event&gt;</code> , page 326

**<window-event>***Open abstract class*

Summary	The base class of events that occur in windows.
Superclasses	<code>&lt;sheet-event&gt;</code>
Init-keywords	<b>region:</b> An instance of type <code>&lt;region&gt;</code> . Required.
Description	The base class of events that occur in windows. Two types of event can occur: <ul style="list-style-type: none"> <li>Changes to the configuration of the window.</li> <li>Changes that require the window to be repainted.</li> </ul> The <b>region:</b> init-keyword specifies a region in which the event occurs.
Operations	The following operation is exported from the <b>DUIM-Sheets</b> module.  <code>event-region</code>
See also	<code>event-region</code> , page 235  <code>&lt;window-configuration-event&gt;</code> , page 324  <code>&lt;window-repaint-event&gt;</code> , page 326

## <window-repaint-event> *Sealed instantiable class*

Summary	The class of events involving repainting of a window.
Superclasses	<window-event>
Init-keywords	None.
Description	The class of events involving repainting of a window.
Operations	None.
See also	<a href="#">handle-repaint</a> , page 253 <a href="#">queue-repaint</a> , page 293 <a href="#">repaint-sheet</a> , page 294 <a href="#">&lt;window-configuration-event&gt;</a> , page 324

## with-brush *Statement macro*

Summary	Executes the supplied code using the specified brush characteristics.
Macro call	<code>with-brush ({medium} #rest {brush-initargs}* ) {body} end</code>
Arguments	<code>medium</code> A Dylan body <sub>bnf</sub> . <code>brush-initargs</code> Dylan arguments <sub>bnf</sub> . <code>body</code> A Dylan body <sub>bnf</sub> .
Values	None.

Description	Executes <i>body</i> using the brush characteristics specified by <i>brush-initargs</i> , and applies the results to <i>medium</i> . The <i>medium</i> specified should be an instance of type < <code>medium</code> >. The <i>brush-initargs</i> can be any valid arguments that specify an instance of < <code>brush</code> >.
See also	<code>with-pen</code> , page 332

**with-clipboard***Statement macro*

Summary	Evaluates a body of code with a clipboard grabbed.	
Signature	<code>with-clipboard (clipboard = sheet) body end</code>	
Arguments	<i>clipboard</i>	A Dylan variable-name <sub>bnf</sub> .
	<i>sheet</i>	A Dylan variable-name <sub>bnf</sub> .
	<i>body</i>	A Dylan body <sub>bnf</sub> .
Values	<i>values</i>	Instances of < <code>object</code> >.
Description	<p>Evaluates <i>body</i> with the clipboard grabbed, returning the results to the clipboard.</p> <p>The macro grabs a lock on the clipboard, using <code>open-clipboard</code>, and then executes <i>body</i>. Once the results of evaluating <i>body</i> have been sent to the clipboard, the clipboard lock is freed using <code>close-clipboard</code>. The <i>clipboard</i> argument is a Dylan variable-name<sub>bnf</sub> used locally in the call to <code>with-clipboard</code>. The <i>sheet</i> argument is a Dylan variable-name<sub>bnf</sub> that evaluates to the sheet associated with <i>clipboard</i>.</p> <p>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.</p>	

You can add more than one format of your data to the clipboard within the scope of this 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.

See also      [`clipboard`](#), page 208

## **with-clipping-region**

*Statement macro*

Summary      Executes the supplied code using the specified clipping region.

Macro call    `with-clipping-region ({medium} {region}) {body} end`

Arguments     *medium*      A Dylan expression<sub>bnf</sub>.

*region*      A Dylan expression<sub>bnf</sub>.

*body*        A Dylan body<sub>bnf</sub>.

Values        None.

Description     Executes *body* using the clipping region specified by *region*, and applies the results to *medium*. The *region* and *medium* expressions should evaluate to instances of `<region>` and `<medium>`, respectively.

## **with-cursor-visible**

*Statement macro*

Summary      Executes the supplied code using the specified cursor settings for a sheet.

Macro call    `with-cursor-visible ({sheet} {visible?}) {body} end`

Arguments     *sheet*        A Dylan expression<sub>bnf</sub>.

<i>visible?</i>	A Dylan expression <sub>bnf</sub> .
<i>body</i>	A Dylan body <sub>bnf</sub> .
Values	None.
Description	<p>Executes <i>body</i> on the specified <i>sheet</i>. If <i>visible?</i> is true, then the pointer cursor associated with <i>sheet</i> is visible throughout the operation. If <i>visible?</i> is false, then the pointer cursor is hidden.</p> <p>The expression <i>sheet</i> should evaluate to an instance of <code>&lt;sheet&gt;</code>. The expression <i>visible?</i> should evaluate to a boolean value.</p>

<b>with-drawing-options</b>		<i>Statement macro</i>
Summary		Runs a body of code in the context of a set of drawing options.
Macro call		<code>with-drawing-options ({medium} #rest {options}*) {body}</code> <code>end</code>
Arguments	<i>medium</i>	A Dylan expression <sub>bnf</sub> .
	<i>options</i>	Dylan arguments <sub>bnf</sub> .
	<i>body</i>	A Dylan body <sub>bnf</sub> .
Values		None.
Description		<p>Runs a body of code in the context of a set of drawing options. The options specified are passed to the function <code>do-with-drawing-options</code> for execution.</p> <p>The <i>medium</i> expression should evaluate to an instance of <code>&lt;medium&gt;</code>.</p>

Note that when using `with-drawing-options` in conjunction with a loop, it is computationally much quicker to use a medium (as shown here) rather than a sheet, and to place the call to `with-drawing-options` outside the loop. If necessary, use `with-sheet-medium` to associate the sheet with the medium, thus:

```
with-sheet-medium (medium = sheet)
  with-drawing-options (medium, brush: color)
    for (x :: <integer> from 0 to 199)
      for (y :: <integer> from 0 to 199)
        draw-point(medium, x, y)
      end
    end
  end
end
```

**Example**

```
with-drawing-options (medium, brush: $red)
  draw-rectangle (medium, 0, 0, 100, 200, filled?: #t)
end;
```

**See also**

`do-with-drawing-options`, page 227

`with-sheet-medium`, page 335

**withdraw-sheet***Generic function*

**Summary** Withdraws the specified sheet from the current display.

**Signature** `withdraw-sheet sheet => ()`

**Arguments** `sheet` An instance of type `<sheet>`.

**Values** None.

**Description** Withdraws the specified sheet from the current display.

**with-frame-manager***Statement macro*

Summary	Executes the supplied code in the context of the specified frame manager.				
Macro call	<code>with-frame-manager ({framem}) {body} end</code>				
Arguments	<table> <tr> <td><i>framem</i></td><td>A Dylan expression<sub>bnf</sub>.</td></tr> <tr> <td><i>body</i></td><td>A Dylan body<sub>bnf</sub>.</td></tr> </table>	<i>framem</i>	A Dylan expression <sub>bnf</sub> .	<i>body</i>	A Dylan body <sub>bnf</sub> .
<i>framem</i>	A Dylan expression <sub>bnf</sub> .				
<i>body</i>	A Dylan body <sub>bnf</sub> .				
Values	None.				
Description	<p>Executes <i>body</i> in the context of <i>framem</i>, by dynamically binding the expression <i>framem</i> to <code>*current-frame-manager*</code>.</p> <p>In practice, you do not need to use <code>with-frame-manager</code> unless you are certain that your code needs to run on a non-primary frame manager.</p> <p>The main place where you need to use this macro is when you call <code>make</code> to create a gadget <i>outside</i> of one of the pane or layout clauses in <code>define frame</code>.</p> <p>Unless you are developing code that needs to run on more than one platform, this is unlikely to be the case, and you can forego use of this macro.</p>				
See also	<code>&lt;frame-manager&gt;</code> , page 243				

**with-identity-transform***Statement macro*

Summary	Executes the supplied code while retaining the current transform.
Macro call	<code>with-identity-transform ({medium}) {body} end</code>
Arguments	<i>medium</i> A Dylan expression <sub>bnf</sub> .

	<i>body</i>	A Dylan body <sub>bnf</sub> .
Values	None.	
Description	<p>Executes <i>body</i> while retaining the current transform for <i>medium</i>.</p> <p>The <i>medium</i> expression should evaluate to an instance of <code>&lt;medium&gt;</code>.</p>	
<b>with-pen</b>		<i>Statement macro</i>
Summary	Executes the supplied code using the specified pen characteristics.	
Macro call	<code>with-pen ({medium} #rest {pen-initargs}* ) {body} end</code>	
Arguments	<p><i>medium</i>      A Dylan expression<sub>bnf</sub>.</p> <p><i>pen-initargs</i>    Dylan arguments<sub>bnf</sub>.</p> <p><i>body</i>          A Dylan body<sub>bnf</sub>.</p>	
Values	None.	
Description	<p>Executes <i>body</i> using the pen characteristics specified by <i>pen-initargs</i>, and applies the results to the expression <i>medium</i>.</p> <p>The <i>medium</i> specified should be an instance of type <code>&lt;medium&gt;</code>. The <i>pen-initargs</i> can be any valid arguments that specify an instance of <code>&lt;pen&gt;</code>.</p>	
See also	<a href="#">with-brush</a> , page 326	

**with-pointer-grabbed***Statement macro*

Summary	Executes a body of code, forwarding all pointer events to a sheet.						
Macro call	<code>with-pointer-grabbed ({sheet} #rest {options}*) {body} end</code>						
Arguments	<table> <tr> <td><i>sheet</i></td><td>A Dylan expression<sub>bnf</sub>.</td></tr> <tr> <td><i>options</i></td><td>Dylan arguments<sub>bnf</sub>.</td></tr> <tr> <td><i>body</i></td><td>A Dylan body<sub>bnf</sub>.</td></tr> </table>	<i>sheet</i>	A Dylan expression <sub>bnf</sub> .	<i>options</i>	Dylan arguments <sub>bnf</sub> .	<i>body</i>	A Dylan body <sub>bnf</sub> .
<i>sheet</i>	A Dylan expression <sub>bnf</sub> .						
<i>options</i>	Dylan arguments <sub>bnf</sub> .						
<i>body</i>	A Dylan body <sub>bnf</sub> .						
Values	None.						
Description	<p>Executes a body of code, forwarding all pointer events to <i>sheet</i>, even if the pointer leaves the sheet-region of <i>sheet</i>. The <i>sheet</i> specified should be an instance of type <code>&lt;sheet&gt;</code>.</p> <p>The macro calls methods for <code>do-with-pointer-grabbed</code>. The code specified by <i>body</i> is used to create a stand-alone method that is used as the code that is run by <code>do-with-pointer-grabbed</code>.</p>						
See also	<code>do-with-pointer-grabbed</code> , page 228						

**with-rotation***Statement macro*

Summary	Executes a body of code with a specified rotation.						
Macro call	<code>with-rotation ({medium} {angle}) {body} end</code>						
Arguments	<table> <tr> <td><i>medium</i></td><td>A Dylan expression<sub>bnf</sub>.</td></tr> <tr> <td><i>angle</i></td><td>A Dylan argument<sub>bnf</sub>.</td></tr> <tr> <td><i>body</i></td><td>A Dylan body<sub>bnf</sub>.</td></tr> </table>	<i>medium</i>	A Dylan expression <sub>bnf</sub> .	<i>angle</i>	A Dylan argument <sub>bnf</sub> .	<i>body</i>	A Dylan body <sub>bnf</sub> .
<i>medium</i>	A Dylan expression <sub>bnf</sub> .						
<i>angle</i>	A Dylan argument <sub>bnf</sub> .						
<i>body</i>	A Dylan body <sub>bnf</sub> .						
Values	None.						

Description	Executes a body of code with a specified rotation. The rotation occurs within the expression <i>medium</i> . This macro calls <code>with-transform</code> to perform the rotation.
	The <i>medium</i> specified should be an instance of type <code>&lt;medium&gt;</code> . The <i>angle</i> should evaluate to an instance of type <code>&lt;real&gt;</code> .
See also	<a href="#">with-scaling</a> , page 334 <a href="#">with-transform</a> , page 337 <a href="#">with-translation</a> , page 337

**with-scaling***Statement macro*

Summary	Executes a body of code with a specified scaling.								
Macro call	<code>with-scaling ({medium} {scale-x} {scale-y}) {body} end</code>								
Arguments	<table> <tr> <td><i>medium</i></td><td>A Dylan expression<sub>bnf</sub>.</td></tr> <tr> <td><i>scale-x</i></td><td>A Dylan argument<sub>bnf</sub>.</td></tr> <tr> <td><i>scale-y</i></td><td>A Dylan argument<sub>bnf</sub>.</td></tr> <tr> <td><i>body</i></td><td>A Dylan body<sub>bnf</sub>.</td></tr> </table>	<i>medium</i>	A Dylan expression <sub>bnf</sub> .	<i>scale-x</i>	A Dylan argument <sub>bnf</sub> .	<i>scale-y</i>	A Dylan argument <sub>bnf</sub> .	<i>body</i>	A Dylan body <sub>bnf</sub> .
<i>medium</i>	A Dylan expression <sub>bnf</sub> .								
<i>scale-x</i>	A Dylan argument <sub>bnf</sub> .								
<i>scale-y</i>	A Dylan argument <sub>bnf</sub> .								
<i>body</i>	A Dylan body <sub>bnf</sub> .								
Values	None.								
Description	Executes a body of code with a specified scaling, denoted by <i>scale-x</i> and <i>scale-y</i> . The scaling occurs within the expression <i>medium</i> . This macro calls <code>with-transform</code> to perform the scaling.								
	The <i>medium</i> specified should be an instance of type <code>&lt;medium&gt;</code> . The <i>scale-x</i> and <i>scale-y</i> should evaluate to an instance of type <code>&lt;real&gt;</code> .								
See also	<a href="#">with-rotation</a> , page 333								

`with-transform`, page 337

`with-translation`, page 337

## with-sheet-medium

*Statement macro*

Summary Associates a sheet with a medium.

Macro call `with-sheet-medium ({medium = sheet}) {body} end`

Arguments `medium` A Dylan name<sub>bnf</sub>.

`sheet` A Dylan expression<sub>bnf</sub>.

`body` A Dylan body<sub>bnf</sub>.

Values None.

Description Associates a sheet with a medium.

Within `body`, the variable `medium` is bound to the medium allocated to `sheet`. The `sheet` specified should be an instance of type `<sheet>`. If `sheet` does not have a medium permanently allocated, one is allocated and associated with `sheet` for the duration of `body`, and then unassociated from `sheet` and deallocated when `body` has been exited. The values of the last form of `body` are returned as the values of `with-sheet-medium`.

The `medium` argument is not evaluated, and must be a symbol that is bound to a medium. The `body` may have zero or more declarations as its first forms.

This macro is a useful way of speeding up drawing operations, since drawing on a sheet requires finding the medium for that sheet. You can use `with-sheet-medium` to associate a known sheet with a medium, and then draw directly onto that medium, as shown in the example.

**Example**

```
with-sheet-medium (medium = sheet)
  with-drawing-options (medium, brush: color)
    for (x :: <integer> from 0 to 199)
      for (y :: <integer> from 0 to 199)
        draw-point(medium, x, y)
      end
    end
  end
end
```

**See also**

[do-with-sheet-medium](#), page 229  
[with-drawing-options](#), page 329

**with-text-style***Statement macro***Summary**

Runs a body of code in the context of a text style.

**Macro call**

```
with-text-style ({medium} #rest {style-initargs}* ) {body}
end
```

**Arguments**

<i>medium</i>	A Dylan expression <sub>bnf</sub> .
<i>style-initargs</i>	Dylan arguments <sub>bnf</sub> .
<i>body</i>	A Dylan body <sub>bnf</sub> .

**Values**

None.

**Description**

Executes *body* using the text style characteristics specified by *style-initargs*, and applies the results to *medium*.

The *medium* specified should be an instance of type `<medium>`. The *style-initargs* can be any valid arguments that specify an instance of `<text-style>`.

Methods for `do-with-text-style` are invoked to run the code.

**See also**

[do-with-text-style](#), page 229

## with-transform

### *Statement macro*

Summary	Executes a body of code with a specified transform.	
Macro call	<code>with-transform ({medium} {transform}) {body} end</code>	
Arguments	<i>medium</i>	A Dylan expression <sub>bnf</sub>
	<i>transform</i>	A Dylan expression <sub>bnf</sub>
	<i>body</i>	A Dylan body <sub>bnf</sub> .
Values	None.	
Description	<p>Executes a body of code with a specified <i>transform</i>. The transform occurs within <i>medium</i>. This macro is used by <code>with-rotation</code>, <code>with-scaling</code>, and <code>with-translation</code>, and calls methods for <code>do-with-transform</code>.</p> <p>The <i>medium</i> specified should be an instance of type <code>&lt;medium&gt;</code>. The <i>transform</i> specified should be an instance of type <code>&lt;transform&gt;</code>.</p>	
See also	<a href="#">do-with-transform</a> , page 230 <a href="#">with-rotation</a> , page 333 <a href="#">with-scaling</a> , page 334 <a href="#">with-translation</a> , page 337	

## with-translation

### *Statement macro*

Summary	Executes a body of code with a specified translation.	
Macro call	<code>with-translation ({medium} {dx} {dy}) {body} end</code>	
Arguments	<i>medium</i>	A Dylan expression <sub>bnf</sub>
	<i>dx</i>	A Dylan argument <sub>bnf</sub> .

	<i>dy</i>	A Dylan argument <sub>bnf</sub> .
	<i>body</i>	A Dylan body <sub>bnf</sub> .
Values	None.	
Description		Executes a body of code with a specified translation, denoted by <i>dx</i> and <i>dy</i> . The translation occurs within <i>medium</i> . This macro calls <code>with-transform</code> to perform the translation.  The <i>medium</i> specified should be an instance of type <code>&lt;medium&gt;</code> . The <i>dx</i> and <i>dy</i> should evaluate to an instance of type <code>&lt;real&gt;</code> .
See also		<code>with-rotation</code> , page 333 <code>with-scaling</code> , page 334 <code>with-transform</code> , page 337

# 6

---

---

# DUIM-Graphics Library

## 6.1 Overview

The DUIM-Graphics library contains interfaces that define a wide variety drawing operations for use in your GUI applications, as well as two classes. The library contains a single module, `duim-graphics`, from which all the interfaces described in this chapter are exposed. Section 6.6 on page 354 contains complete reference entries for each exposed interface.

The DUIM graphic drawing model is an idealized model of graphical pictures. The model provides the language that application programs use to describe the intended visual appearance of textual and graphical output. Usually not all of the contents of the screen are described using the graphic drawing model. For example, menus and scroll bars would usually be described in higher-level terms.

An important aspect of the DUIM graphic drawing model is its extreme device independence. The model describes ideal graphical images and ignores limitations of actual graphics devices. One consequence of this is that the actual visual appearance of the screen can only be an approximation of the appearance specified by the model: however, another important consequence of this is that the model is highly portable.

DUIM separates output into two layers:

1. A text/graphics layer in which you specify the desired visual appearance independent of device resolution and characteristics
2. A rendering layer in which some approximation of the desired visual appearance is created on the device.

Of course application programs can inquire about the device resolution and characteristics if they wish and modify their desired visual appearance on that basis. There is also a third layer above these two layers, the adaptive toolkit layer where one specifies the desired functionality rather than the desired visual appearance.

## 6.2 Definitions

This section contains definitions of terms that will be used in this chapter.

**Drawing plane** A drawing plane is an infinite two-dimensional plane on which graphical output occurs. The drawing plane contains an arrangement of colors and opacities that is modified by each graphical output operation. It is not possible to read back the contents of a drawing plane, except by examining the output-history. Normally each window has its own drawing plane.

**Coordinates** Coordinates are a pair of real numbers in implementation-defined units that identify a point in the drawing plane.

**Mediums** In this chapter, we use a medium as a destination for output. The medium has a drawing plane, two designs (called the medium's foreground and background), a transformation, a clipping region, a line style, and a text style. There are per-medium, dynamically scoped, default drawing options. Different medium classes are provided to allow you to draw on different sorts of devices, such as displays, printers, and virtual devices such as bitmaps.

<b>Sheets</b>	Many sheets can be used for doing output, so the drawing functions can also take a sheet as the output argument. In this case, drawing function “trampolines” to the sheet’s medium. So, while the functions defined here are specified to be called on mediums, they can also be called on sheets.
<b>Streams</b>	<p>A stream is a special kind of sheet that implements the stream protocol; streams include additional state such as the current text cursor (which is some point in the drawing plane).</p> <p>By default, the “fundamental” coordinate system of a DUIM stream (not a general sheet or medium, whose fundamental coordinate system is not defined) is a left handed system with x increasing to the right, and y increasing downward. (0,0) is at the upper left corner.</p> <p>For more general information about DUIM streams, you should refer to the manual <i>Library Reference: System and I/O</i>.</p>

## 6.3 Drawing is approximate

Note that although the drawing plane contains an infinite number of mathematical points, and drawing can be described as an infinite number of color and opacity computations, the drawing plane cannot be viewed directly and has no material existence: it is only an abstraction. What *can* be viewed directly is the result of rendering portions of the drawing plane onto a medium. No infinite computations or objects of infinite size are required to implement DUIM, because the results of rendering have finite size and finite resolution.

A drawing plane is described as having infinitely fine spatial, color, and opacity resolution, and as allowing coordinates of unbounded positive or negative magnitude. A viewport into a drawing plane, on the other hand, views only a finite region (usually rectangular) of the drawing plane. Furthermore, a viewport has limited spatial resolution and can only produce a limited number of

colors. These limitations are imposed by the display hardware on which the viewport is displayed. A viewport also has limited opacity resolution, determined by the finite arithmetic used in the drawing engine.

Coordinates are real numbers in implementation-defined units. Often these units equal the spatial resolution of a viewport, so that a line of thickness 1 is equivalent to the thinnest visible line. However, this equivalence is not required and should not be assumed by application programs.

DUIM can be quite restrictive in the size and resolution of its viewports. For example, the spatial resolution might be only a few dozen points per inch, the maximum size might be only a few hundred points on a side, and there could be as few as two displayable colors (usually black and white). Fully transparent and fully opaque opacity levels are supported, but a DUIM implementation might support only a few opacity levels in between (or possibly even none). A DUIM implementation might implement color blending and unsaturated colors by stippling, although it is preferred, when possible, for a viewport to display a uniform color as a uniform color rather than as a perceptible stipple.

However, there are no such limitations when DUIM records the output to a sheet, since DUIM just remembers the drawing operations that were performed, not the results of rendering.

The application programmer uses the DUIM graphic drawing model as an interface to describe the intended visual appearance. DUIM then approximates that ideal appearance in a viewport, within its limitations of spatial resolution, color resolution, number of simultaneously displayable colors, and drawing speed.

Naturally, doing this usually requires trade-offs, for example between speed and accuracy, and these trade-offs depend on the hardware and software environment and the user concerns in any given situation. For example:

- If the device only supports a limited number of colors, the desired color may be approximated using techniques such as dithering or stippling.
- If the device cannot draw curves precisely, they may be approximated, with or without anti-aliasing.

- If the device has limited opacity resolution, color blending may be approximate. A viewport might display colors that do not appear in the drawing plane, both because of color and opacity approximation and because of anti-aliasing at the edges of drawn shapes.

Drawing computations are always carried out “in color”, even if the viewport is only capable of displaying black and white. In other words, the DUIM drawing model is always the fully general model, even if an implementation’s color resolution is limited enough that full use of the model is not possible. Of course an application that fundamentally depends on color will not work well on a viewport that cannot display color. Other applications will degrade gracefully.

Whether the implementation uses raster graphics or some other display technique is invisible at this interface. DUIM does not specify the existence of pixels nor the exact details of scan conversion, which will vary from one drawing engine to the next.

## 6.4 Rendering conventions for geometric shapes

This section describes the conventions for how DUIM renders a shape on a display device.

When DUIM draws a geometric shape on a display device, the idealized geometric shape must somehow be rendered on that device. This involves mapping points on the idealized geometric shape onto points on the display device.

Idealized geometric shapes are made up of a set of mathematical points which have no size. The rendering of these shapes on the display device is usually composed of pixels, which are roughly square, and are specified in “device coordinates”. Device coordinates are calculated by transforming the user-supplied coordinates by each of the following:

- The user-supplied transformation
- The medium transformation
- The transformation that maps from the sheet to the display device

**Note:** If the last of these is a pure translation that translates by an integer multiple of device units, then it has no effect on the rendering other than placement of the figure drawn on the display device.

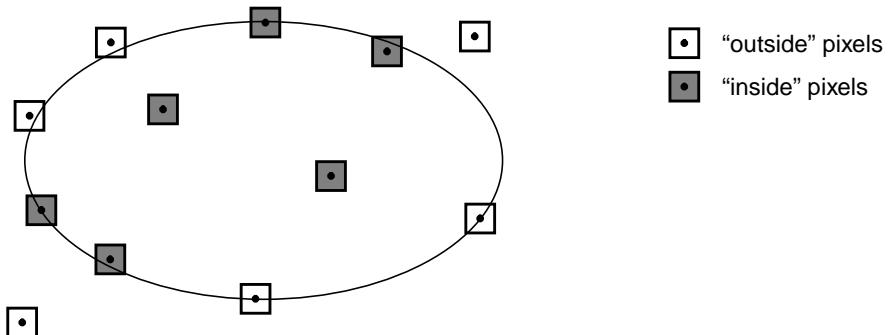
Roughly speaking, a pixel is affected by drawing a shape only when it is inside that shape. Since pixels are little squares, and the abstract points in an idealized geometric shape have no size, most shapes will have many pixels that lie only partially inside the shape. It is important, therefore, to describe which pixels will be affected when rendering a shape, and which will not.

On devices that support color or grayscale, the rendering engine uses anti-aliasing techniques to render pixels that lie only partially inside the shape. That is, the affected pixels are drawn a little lighter than pixels that are wholly within the shape, the precise shade depending on how much of it is inside the shape.

The conventions used by DUIM are the same as the conventions used by X11:

- A pixel is addressed by its upper-left corner.
- A pixel is considered to be inside a shape, and hence affected by the rendering of that shape, if the center of the pixel is inside the shape. If the center of the pixel lies exactly on the boundary of the shape, it is considered to be inside the shape if the inside of the shape is immediately to the right of the center point of the pixel (that is, an increasing x direction on the display device). If the center of the pixel lies exactly on a horizontal boundary, it is considered to be inside the shape if the inside of the shape is immediately below the center point of the pixel (that is, an increasing y direction on the display device). This situation is illustrated in Figure 6.1.
- An unfilled idealized geometric shape is drawn by calculating an artificial area for the shape, and then deciding which pixels are inside or outside that area, using the rules described above. The artificial area is calculated by taking the filled shape consisting of those points that are within half the line thickness from the outline curve (using a normal distance function, that is, the length of the line drawn at right angles to

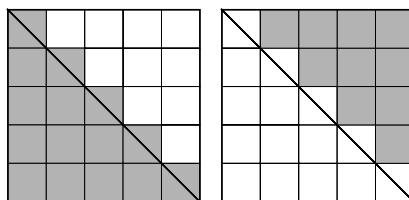
the tangent to the outline curve at the nearest point). To visualize this, imagine a filled shape the same size as the unfilled shape, and overlay on this filled shape an identical, but slightly smaller, unfilled shape.



**Figure 6.1** How pixels are defined to be “inside” and “outside” shapes

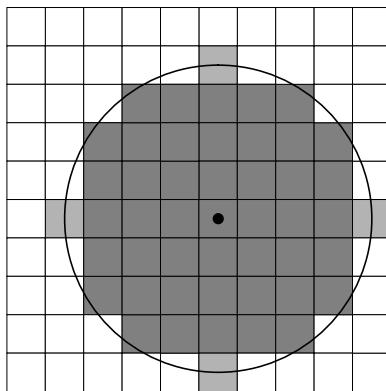
It is important to note that these rules imply that the decision point used for insideness checking is offset from the point used for addressing the pixel by half a device unit in both the x and y directions. It is worth considering the motivations for these conventions.

When two shapes share a common edge, it is important that only one of the shapes own any pixel. The two triangles in Figure 6.2 illustrate this. The pixels along the diagonal belong to the lower figure. When the decision point of the pixel (its center) lies to one side of the line or the other, there is no issue. When the boundary passes through a decision point, which side the inside of the figure is on is used to decide.



**Figure 6.2** Two triangles

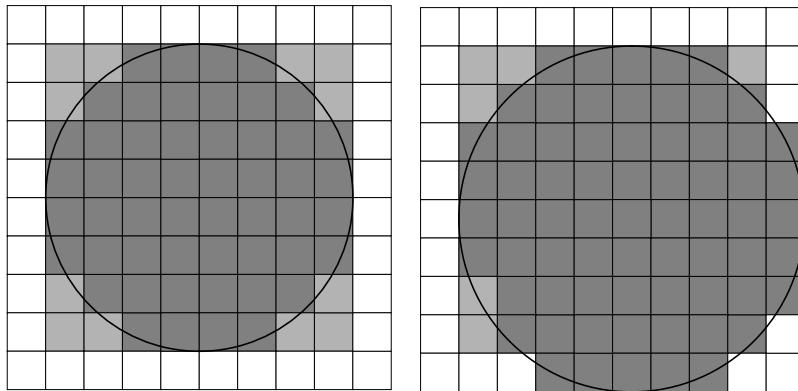
The reason for choosing the decision point half a pixel offset from the address point is to reduce the number of common figures (such as rectilinear lines and rectangles with integral coordinates) that invoke the boundary condition rule. This usually leads to more symmetrical results. For instance, shows a circle drawn when the decision point is the same as the address point. The four lighter points are indeterminate: it is not clear whether they are inside or outside the shape. Since each boundary case is determined according to which side has the figure on it, and since the same rule must be applied uniformly for all figures, there is no choice but to pick only two of the four points, leading to an undesirable lopsided figure.



**Figure 6.3** Choosing any two of the shaded pixels causes asymmetry

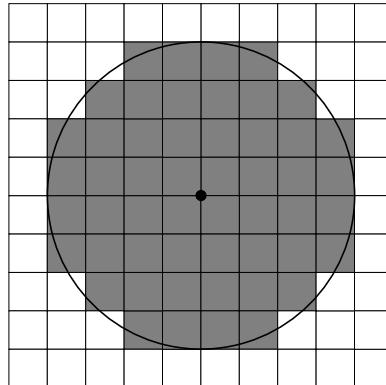
If all four boundary points had been chosen instead, the result would be a symmetrical figure. However, since this figure is symmetrical about a whole pixel, it is one pixel wider than it ought to be. The problem with this can be

seen clearly in Figure 6.4, in which a circle is drawn over a square. In the left-hand figure, the decision point is at the center of the pixel, but in the right-hand figure, it is not.



**Figure 6.4** Two forms of a circle inscribed in a square

It is for this reason that the decision point is at the center of the pixel. This draws circles that look like the one in Figure 6.5.

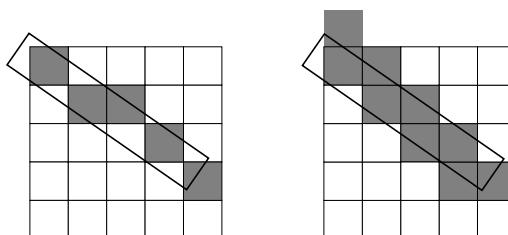


**Figure 6.5** An aesthetically pleasing circle

A consequence of these rendering conventions is that, when the start or end coordinate (minus half the line thickness, if the shape is a path) is not an integer, then rendering is not symmetric under reflection transformations. Thus, to correctly and portably draw an outline of thickness 1 around a (rectilinear) rectangular area with integral coordinates, the outline path must have half-integral coordinates. Drawing rectilinear areas whose boundaries are not on pixel boundaries cannot be guaranteed to be portable. In other words, the “control points” for a rectangular area are at the corners, while the control points for a rectilinear path are in the center of the path, not at the corners. Therefore, in order for a path and an area to abut seamlessly, the coordinates of the path must be offset from the coordinates of the area by half the thickness of the path.

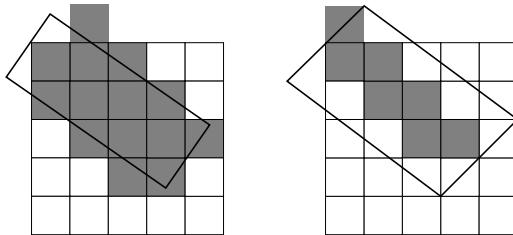
#### 6.4.1 Permissible alternatives during rendering

Some platforms may distinguish between lines of the minimum thinness from lines that are thicker than that. The two rasterizations depicted in Figure 6.6 are both perfectly reasonable rasterizations of tilted lines that are a single device unit wide. The right-hand line is drawn as a tilted rectangle, the left as the “thinnest visible” line.



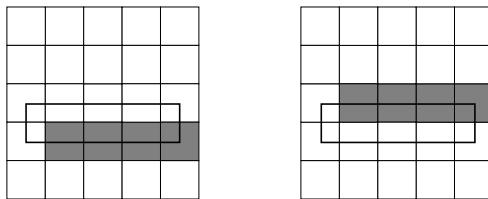
**Figure 6.6** Two examples of lines of thickness 1

For thick lines, a platform may choose to draw the exact tilted fractional rectangle, or the coordinates of that rectangle might be rounded so that it is distorted into another polygonal shape. The latter case may be prove to be faster on some platforms. The two rasterizations depicted in Figure 6.7 are both reasonable.



**Figure 6.7** Two examples of lines of thickness 2

The decision about which side of the shape to take when a boundary line passes through the decision point is made arbitrarily, although this is compatible with the X11 definition. This is not necessarily the most convenient decision. The main problem with this is illustrated by the case of a horizontal line (see Figure 6.8). The DUIM definition draws the rectangular slice above the coordinates, since those pixels are the ones whose centers have the figure immediately above them. This definition makes it simpler to draw rectilinear borders around rectilinear areas.



**Figure 6.8** Two possible definitions of horizontal lines. Left figure is X11 definition

## 6.5 Drawing using path related functions

A number of functions are provided that let you perform a number of connected drawing operations by encapsulating all the operations as a single path, rendering the graphic itself only when the whole path has been defined explicitly. You can use these functions by following the general procedure below:

1. Create a new path using `start-path`.
2. Define the appearance of the path using any combination of `line-to`, `move-to`, `curve-to`, and `arc-to`.
3. Optionally, use `close-path` to create a closed path from the segments defined in step 2 above.
4. End the current path definition using `end-path` (if you have not already used `close-path`).
5. Render the outline of the path to the drawable object using `stroke-path`.
6. If the path you created is closed, flood fill the path using `fill-path`.

Each of these functions is described in a little more in the following sections. For full details about each individual function, refer to its full reference entry in Section 6.6.

### 6.5.1 Functions for controlling the definition of a path

The following generic functions provide overall control of the definition of a path. In each case, the argument `drawable` is either a sheet or a medium.

<code>start-path</code>	<i>Generic function</i>
-------------------------	-------------------------

```
start-path drawable => ()
```

Starts a new path on `drawable`. The path can be created with any number of calls to `line-to`, `curve-to`, `arc-to`, and `move-to`. Its appearance can also be manipulated using `fill-path` and `stroke-path`.

After creating the path, use either `close-path` or `end-path` to finish the path, or `abort-path` to abandon it altogether.

**end-path***Generic function***end-path** *drawable* => ()

Ends the definition of the current path in *drawable*. Once the definition has been ended, the path can be rendered to the drawable using **fill-path** or **stroke-path**.

The function **close-path** can also be used to end the definition of a path.

**close-path***Generic function***close-path** *drawable* => ()

Closes the current path on the *drawable*: that is, creates a closed figure from the elements already defined.

For example, if you create a path that has four connected lines (using **line-to**), you can use **close-path** to join the first and last lines in the path to create a closed, five-sided figure.

**abort-path***Generic function***abort-path** *drawable* => ()

Aborts the current path on *drawable*. Any operations that have been performed since the last call to **start-path** are discarded.

**fill-path***Generic function***fill-path** *drawable* => ()

Uses the current brush to fill the current path on *drawable*. Only closed paths can be filled. If the path has not already been closed using **close-path**, it is closed automatically.

**stroke-path** *Generic function*

```
stroke-path drawable => ()
```

Uses the current pen to draw the current path on *drawable*. Note that the path must not have been previously filled. This function does not close the path: you must use `close-path` if you wish to do this.

### 6.5.2 Functions for describing the appearance of a path

The following generic functions actually perform drawing operations within a path. Again, in each case, the argument *drawable* is either a sheet or a medium. All other arguments are instances of `<real>`.

**line-to** *Generic function*

```
line-to drawable x y => ()
```

Draws a line from the current position in the path to (*x,y*).

**curve-to** *Generic function*

```
curve-to drawable x1 y1 x2 y2 x3 y3 => ()
```

Draws a curve in the current path on *drawable* starting from the current position, and passing through (*x1,y1*), (*x2, y2*), and (*x3, y3*).

**move-to** *Generic function*

```
move-to drawable x y => ()
```

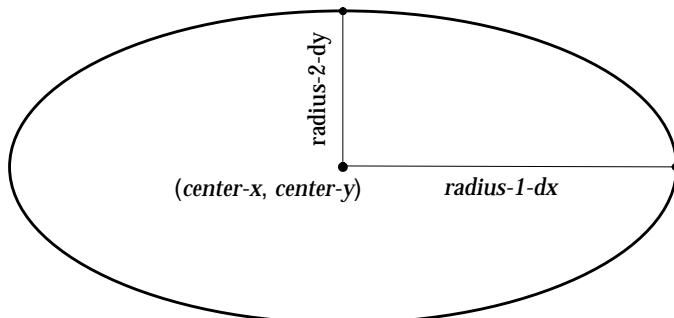
Move the position in the current path on *drawable* to (*x,y*).

The function `move-to` can be used several times within the definition of a path, allowing for the definition of several visually separate sections within the same path.

**arc-to****Generic function**

```
arc-to drawable center-x center-y radius-1-dx radius-1-dy radius-2-dx radius-2-dy
#key start-angle end-angle => ()
```

Draws an arc in the current path on *drawable*.



**Figure 6.9** Description of the arguments for arc-to

The center of the arc is defined by (*center-x*, *center-y*), the points furthest away from the center for each radius are calculated by adding *radius-1-dx* and *radius-1-dy* to *center-x* and *center-y* respectively (to calculate the outermost points for the first radius), and adding *radius-2-dx* and *radius-2-dy* to *center-x* and *center-y* respectively (to calculate the outermost points for the second radius).

The arguments *start-angle* and *end-angle* define the extent of the arc that is drawn.

For each function listed above, an equivalent function is also provided that passes composite objects in its arguments, rather than separate coordinates. These functions take the same name as the functions above, but with a \* character appended. (Thus, `line-to*` performs the same operation as `line-to`, but passes composite objects in its arguments). You should be aware that using these composite object functions may lead to a loss of performance. For more details, see the full reference entries for each function.

## 6.6 DUIM-Graphics Module

This section contains a complete reference of all the interfaces that are exported from the `duim-graphics` module.

### **abort-path** *Generic function*

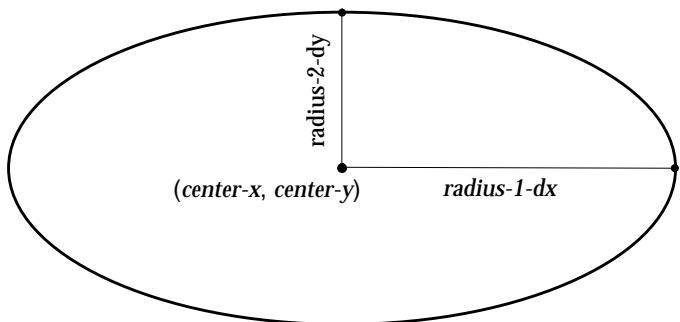
Summary	Aborts the current path on the specified drawable object.
Signature	<code>abort-path <i>drawable</i> =&gt; ()</code>
Arguments	<code>drawable</code> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
Values	None.
Description	Aborts the current path on <i>drawable</i> . Any operations that have been performed since the last call to <code>start-path</code> are discarded.
See also	<a href="#">close-path</a> , page 356 <a href="#">end-path</a> , page 386 <a href="#">start-path</a> , page 392

### **arc-to** *Generic function*

Summary	Draws an arc in the current path on the specified drawable.
Signature	<code>arc-to <i>drawable center-x center-y radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key start-angle end-angle</i> =&gt; ()</code> <code>arc-to* <i>drawable center radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key start-angle end-angle</i> =&gt; ()</code>

Arguments	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	<i>radius-1-dx</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>radius-1-dy</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>radius-2-dx</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>radius-2-dy</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>start-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .
	<i>end-angle</i>	An instance of type <code>false-or(&lt;real&gt;)</code> .
	The following arguments are specific to <code>arc-to</code> .	
	<i>center-x</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>center-y</i>	An instance of type <code>&lt;real&gt;</code> .
	The following argument is specific to <code>arc-to*</code> .	
	<i>center</i>	An instance of type <code>&lt;transform&gt;</code> .
Values	None.	
Description	<p>Draws an arc in the current path on the specified <code>drawable</code>. This function is used, in combination with <code>line-to</code>, <code>curve-to</code>, and <code>move-to</code>, to define a path. The function <code>start-path</code> should be used to start the definition of the path, and <code>end-path</code> can be used to finish the definition.</p> <p>The center of the arc is defined by (<code>center-x</code>, <code>center-y</code>), and the extreme points of the virtual ellipse around the arc (that is, the points furthest away from the center for each radius) are calculated by adding the radius vectors <code>radius-1-dx</code> and <code>radius-1-dy</code> to <code>center-x</code> and <code>center-y</code> respectively (to calculate the outermost points for the first radius), and adding the radius vectors <code>radius-2-dx</code> and <code>radius-2-dy</code> to <code>center-x</code> and <code>center-y</code> respectively (to calculate the outermost points for the second radius).</p>	

Please note that `arc-to` does not currently support arcs whose orientation is not axis-aligned ellipses. For all practical purposes, this means that `radius-1-dy` and `radius-2-dx` must always be 0.



The arguments `start-angle` and `end-angle` define the extent of the arc that is drawn.

The function `arc-to*` is identical to `arc-to`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

#### See also

- `curve-to`, page 360
- `draw-bezier-curve`, page 364
- `draw-line`, page 370
- `line-to`, page 387
- `move-to`, page 389

## close-path

*Generic function*

#### Summary

Closes the current path on the specified drawable.

#### Signature

`close-path drawable => ()`

Arguments	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
Values	None.	
Description	Closes the current path on the <i>drawable</i> : that is, creates a closed figure from the elements already defined.  For example, if you create a path that has four connected lines (using <code>line-to</code> ), you can use <code>close-path</code> to join the first and last lines in the path to create a closed, five-sided figure.  Only closed paths can be filled, although <code>fill-path</code> will close a non-closed path automatically.	
See also	<code>abort-path</code> , page 354  <code>end-path</code> , page 386  <code>start-path</code> , page 392	

<b>copy-area</b>	<i>Generic function</i>														
Summary	Copies a rectangle of pixels from a specified medium to the same medium.														
Signature	<code>copy-area medium from-x from-y width height to-x to-y #key function =&gt; ()</code>														
Arguments	<table> <tr> <td><i>medium</i></td> <td>An instance of type <code>&lt;medium&gt;</code>.</td> </tr> <tr> <td><i>from-x</i></td> <td>An instance of type <code>&lt;coordinate&gt;</code>.</td> </tr> <tr> <td><i>from-y</i></td> <td>An instance of type <code>&lt;coordinate&gt;</code>.</td> </tr> <tr> <td><i>width</i></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> <tr> <td><i>height</i></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> <tr> <td><i>to-x</i></td> <td>An instance of type <code>&lt;coordinate&gt;</code>.</td> </tr> <tr> <td><i>to-y</i></td> <td>An instance of type <code>&lt;coordinate&gt;</code>.</td> </tr> </table>	<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .	<i>from-x</i>	An instance of type <code>&lt;coordinate&gt;</code> .	<i>from-y</i>	An instance of type <code>&lt;coordinate&gt;</code> .	<i>width</i>	An instance of type <code>&lt;integer&gt;</code> .	<i>height</i>	An instance of type <code>&lt;integer&gt;</code> .	<i>to-x</i>	An instance of type <code>&lt;coordinate&gt;</code> .	<i>to-y</i>	An instance of type <code>&lt;coordinate&gt;</code> .
<i>medium</i>	An instance of type <code>&lt;medium&gt;</code> .														
<i>from-x</i>	An instance of type <code>&lt;coordinate&gt;</code> .														
<i>from-y</i>	An instance of type <code>&lt;coordinate&gt;</code> .														
<i>width</i>	An instance of type <code>&lt;integer&gt;</code> .														
<i>height</i>	An instance of type <code>&lt;integer&gt;</code> .														
<i>to-x</i>	An instance of type <code>&lt;coordinate&gt;</code> .														
<i>to-y</i>	An instance of type <code>&lt;coordinate&gt;</code> .														

	<i>function</i>	An instance of type <function>. Default value: \$bool-e-1.
Values	None	
Description	Copies the pixels from the <i>medium</i> starting at the position specified by ( <i>from-x</i> , <i>from-y</i> ) to the position ( <i>to-x</i> , <i>to-y</i> ) on the same medium. A rectangle whose width and height is specified by <i>width</i> and <i>height</i> is copied. If <i>medium</i> is a medium or a stream, then the x and y values are transformed by the user transformation. The copying must be done by <code>medium-copy-copy</code> .	
See also	<code>copy-from-pixmap</code> , page 358 <code>copy-to-pixmap</code> , page 359	

**copy-from-pixmap***Generic function*

Summary	Copies a rectangle of pixels from the specified pixmap to the specified medium.	
Signature	<code>copy-from-pixmap pixmap pixmap-x pixmap-y width height medium medium-x medium-y #key function =&gt; ()</code>	
Arguments	<i>pixmap</i>	An instance of type <pixmap>.
	<i>pixmap-x</i>	An instance of type <coordinate>.
	<i>pixmap-y</i>	An instance of type <coordinate>.
	<i>width</i>	An instance of type <integer>.
	<i>height</i>	An instance of type <integer>.
	<i>medium</i>	An instance of type <coordinate>.
	<i>medium-x</i>	An instance of type <coordinate>.
	<i>medium-y</i>	An instance of type <coordinate>.

<i>function</i>	An instance of type <function>. Default value: \$bool-e-1.
Values	None
Description	Copies a rectangle of pixels from <i>pixmap</i> starting at the position specified by ( <i>pixmap-x</i> , <i>pixmap-y</i> ) into <i>medium</i> at the position ( <i>medium-x</i> , <i>medium-y</i> ). A rectangle whose width and height is specified by <i>width</i> and <i>height</i> is copied. If <i>medium</i> is a medium or a stream, then <i>medium-x</i> and <i>medium-y</i> are transformed by the user transformation. The copying must be done by <code>medium-copy-copy</code> .
See also	<code>copy-area</code> , page 357 <code>copy-to-pixmap</code> , page 359 < <code> pixmap</code> >, page 390

## copy-to-pixmap *Generic function*

Summary	Copies a rectangle of pixels from the specified medium to the specified <i>pixmap</i> .	
Signature	<code>copy-to-pixmap <i>medium</i> <i>medium-x</i> <i>medium-y</i> <i>width</i> <i>height</i> <i>pixmap</i> <i>pixmap-x</i> <i>pixmap-y</i> #key <i>function</i> =&gt; ()</code>	
Arguments	<i>medium</i>	An instance of type < <code>medium</code> >.
	<i>medium-x</i>	An instance of type < <code>coordinate</code> >.
	<i>medium-y</i>	An instance of type < <code>coordinate</code> >.
	<i>width</i>	An instance of type < <code>integer</code> >.
	<i>height</i>	An instance of type < <code>integer</code> >.
	<i>pixmap</i>	An instance of type < <code> pixmap</code> >.
	<i>pixmap-x</i>	An instance of type < <code>coordinate</code> >.

<i>pixmap-y</i>	An instance of type <coordinate>.
<i>function</i>	An instance of type <function>. Default value: \$bool-e-1.
Values	None
Description	<p>Copies the pixels from the <i>medium</i> starting at the position specified by (<i>medium-x</i>,<i>medium-y</i>) into <i>pixmap</i> at the position specified by (<i>pixmap-x</i>,<i>pixmap-y</i>). A rectangle whose width and height is specified by <i>width</i> and <i>height</i> is copied. If <i>medium</i> is a medium or a stream, then <i>medium-x</i> and <i>medium-y</i> are transformed by the user transformation. The copying must be done by <i>medium-copy-copy</i>.</p> <p>If <i>pixmap</i> is not supplied, a new pixmap will be allocated.</p>
See also	<p><a href="#">copy-area</a>, page 357</p> <p><a href="#">copy-from-pixmap</a>, page 358</p>

<b>curve-to</b>		<i>Generic function</i>
Summary		Draws a curve through three specified points in the current path on the specified drawable.
Signature		<pre>curve-to  <i>drawable</i> <i>x1</i> <i>y1</i> <i>x2</i> <i>y2</i> <i>x3</i> <i>y3</i> =&gt; ()</pre> <pre>curve-to* <i>drawable</i> <i>point1</i> <i>point2</i> <i>point3</i> =&gt; ()</pre>
Arguments	<i>drawable</i>	An instance of type <i>type-union(&lt;sheet&gt;, &lt;medium&gt;)</i> .
The following arguments are specific to <b>curve-to</b> .		
<i>x1</i>	An instance of type <real>.	
<i>y1</i>	An instance of type <real>.	
<i>x2</i>	An instance of type <real>.	

<i>y2</i>	An instance of type <real>.
<i>x3</i>	An instance of type <real>.
<i>y3</i>	An instance of type <real>.
The following arguments are specific to <code>curve-to*</code> .	
<i>point1</i>	An instance of type <transform>.
<i>point2</i>	An instance of type <transform>.
<i>point3</i>	An instance of type <transform>.
Values	None.
Description	<p>Draws a curve in the current path on <i>drawable</i> starting from the current position, and passing through (<i>x1</i>,<i>y1</i>), (<i>x2</i>, <i>y2</i>), and (<i>x3</i>, <i>y3</i>).</p> <p>This function is used, in combination with <code>line-to</code>, <code>move-to</code>, and <code>arc-to</code>, to define a path. The function <code>start-path</code> should be used to start the definition of the path, and <code>end-path</code> can be used to finish the definition.</p> <p>The function <code>curve-to*</code> is identical to <code>curve-to</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>
See also	<p><code>arc-to</code>, page 354</p> <p><code>draw-bezier-curve</code>, page 364</p> <p><code>draw-line</code>, page 370</p> <p><code>line-to</code>, page 387</p> <p><code>move-to</code>, page 389</p>

**destroy-pixmap***Generic function*

Summary      Destroys the specified pixmap.

Signature	<code>destroy-pixmap pixmap =&gt; ()</code>
Arguments	<code>pixmap</code> An instance of type <code>&lt;pixmap&gt;</code> .
Values	None
Description	Destroys <i>pixmap</i> .
See also	<code>draw-pixmap</code> , page 373

**do-with-output-to-pixmap***Generic function*

Summary	Returns a pixmap for the specified medium.										
Signature	<code>do-with-output-to-pixmap medium continuation #key width height clear? =&gt; pixmap</code>										
Arguments	<table> <tr> <td><code>medium</code></td> <td>An instance of type <code>&lt;medium&gt;</code>.</td> </tr> <tr> <td><code>continuation</code></td> <td>An instance of type <code>&lt;function&gt;</code>.</td> </tr> <tr> <td><code>width</code></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> <tr> <td><code>height</code></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> <tr> <td><code>clear?</code></td> <td>An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</td> </tr> </table>	<code>medium</code>	An instance of type <code>&lt;medium&gt;</code> .	<code>continuation</code>	An instance of type <code>&lt;function&gt;</code> .	<code>width</code>	An instance of type <code>&lt;integer&gt;</code> .	<code>height</code>	An instance of type <code>&lt;integer&gt;</code> .	<code>clear?</code>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
<code>medium</code>	An instance of type <code>&lt;medium&gt;</code> .										
<code>continuation</code>	An instance of type <code>&lt;function&gt;</code> .										
<code>width</code>	An instance of type <code>&lt;integer&gt;</code> .										
<code>height</code>	An instance of type <code>&lt;integer&gt;</code> .										
<code>clear?</code>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .										
Values	<code>pixmap</code> An instance of type <code>&lt;pixmap&gt;</code> .										
Description	<p>Returns a pixmap for the specified medium. This function is called by <code>with-output-to-pixmap</code> and returns the pixmap that is operated on. If you are subclassing <code>&lt;medium&gt;</code>, you must define new methods on this function.</p> <p>The <i>width</i> and <i>height</i> are integers that give the width and height of the pixmap. If they are unsupplied, the result pixmap will be large enough to contain all of the output done by the body of code executed by <code>with-output-to-pixmap</code>.</p>										

See also [with-output-to-pixmap](#), page 393

## **draw-arrow** *Generic function*

**Summary** Draws an arrow between two specified points.

**Signature**

```
draw-arrow drawable x1 y1 x2 y2 #key from-head? to-head? head-length head-width => ()
```

```
draw-arrow* drawable point1 point2 #key from-head? to-head? head-length head-width => ()
```

**Arguments**

<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
<i>from-head?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#f</code> .
<i>to-head?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
<i>head-length</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: 10.
<i>head-width</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: 5.

The following arguments are specific to `draw-arrow`.

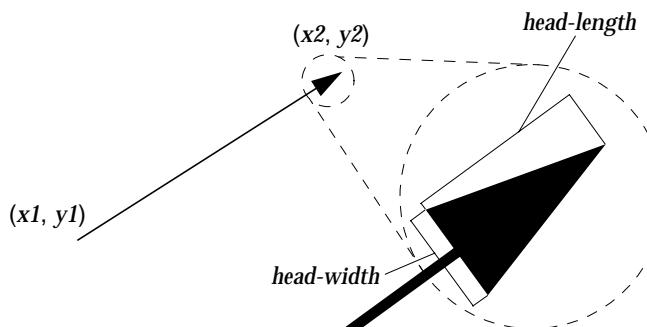
<i>x1</i>	An instance of type <code>&lt;real&gt;</code> .
<i>y1</i>	An instance of type <code>&lt;real&gt;</code> .
<i>x2</i>	An instance of type <code>&lt;real&gt;</code> .
<i>y2</i>	An instance of type <code>&lt;real&gt;</code> .

The following arguments are specific to `draw-arrow*`.

<i>point1</i>	An instance of type <code>&lt;transform&gt;</code> .
<i>point2</i>	An instance of type <code>&lt;transform&gt;</code> .

**Values** None.

Description	<p>Draws an arrow on <i>drawable</i> between two <math>(x_1, y_1)</math> and <math>(x_2, y_2)</math>, using the current pen. Dashed lines start dashing from the first point.</p> <p>If <i>from-head?</i> is <code>#t</code>, then the arrow-head points from <math>(x_1, y_1)</math> to <math>(x_2, y_2)</math>. If <i>to-head?</i> is <code>#t</code>, then the arrow-head points from <math>(x_2, y_2)</math> to <math>(x_1, y_1)</math>.</p> <p>If both <i>from-head?</i> and <i>to-head?</i> are <code>#t</code>, then a double-headed arrow is drawn.</p> <p>The arguments <i>head-length</i> and <i>head-width</i> specify the length and width of the arrow-head respectively, in pixels.</p>
-------------	---



The function `draw-arrow*` is identical to `draw-arrow`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also      `draw-line`, page 370

## draw-bezier-curve

## Generic function

Summary      Draws a bezier curve through the specified set of points.

Signature      `draw-bezier-curve sheet coord-seq #key filled? => ()`  
`draw-bezier-curve* drawable points #key filled? => ()`

Arguments	<i>filled?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
The following arguments are specific to <code>draw-bezier-curve</code> .		
	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .
	<i>coord-seq</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;coordinate&gt;)</code> .
The following arguments are specific to <code>draw-bezier-curve*</code> .		
	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	<i>points</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;point&gt;)</code> .
Values	None.	
Description	<p>Draws a bezier curve on <i>sheet</i> or <i>drawable</i> (depending on the function you use) through the sequence of coordinates given by <i>coord-seq</i>, using the current pen. Dashed lines start dashed from the first point.</p> <p>If <i>filled?</i> is <code>#t</code> then the bezier-curve will be filled, using the current brush.</p> <p>The function <code>draw-bezier-curve*</code> is identical to <code>draw-bezier-curve</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>	
See also	<p><code>curve-to</code>, page 360</p> <p><code>draw-line</code>, page 370</p>	

<b>draw-circle</b>	<i>Generic function</i>
Summary	Draws a circle with the specified center and radius.
Signature	<code>draw-circle <i>drawable</i> <i>center-x</i> <i>center-y</i> <i>radius</i> #key <i>start-angle</i> <i>end-angle</i> <i>filled?</i> =&gt; ()</code> <code>draw-circle* <i>drawable</i> <i>center</i> <i>radius</i> #key <i>start-angle</i> <i>end-angle</i> <i>filled?</i> =&gt; ()</code>
Arguments	<p><i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</p> <p><i>radius</i> An instance of type <code>&lt;real&gt;</code>.</p> <p><i>start-angle</i> An instance of type <code>false-or(&lt;real&gt;)</code>.</p> <p><i>end-angle</i> An instance of type <code>false-or(&lt;real&gt;)</code>.</p> <p><i>filled?</i> An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</p>
	The following arguments are specific to <code>draw-circle</code> .
	<p><i>center-x</i> An instance of type <code>&lt;real&gt;</code>.</p> <p><i>center-y</i> An instance of type <code>&lt;real&gt;</code>.</p>
	The following argument is specific to <code>draw-circle*</code> .
	<p><i>center</i> An instance of type <code>&lt;transform&gt;</code>.</p>
Values	None.
Description	<p>Draws a circle on <i>drawable</i> with center (<i>center-x</i>,<i>center-y</i>) and a radius of <i>radius</i> pixels, using the current pen.</p> <p>The <i>start-angle</i> and <i>end-angle</i> arguments let you draw a sector of a circle rather than a whole circle.</p> <p>If <i>filled?</i> is <code>#t</code>, then the circle will be filled, using the current brush.</p>

The function `draw-circle*` is identical to `draw-circle`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also      `draw-ellipse`, page 367  
                 `draw-oval`, page 372

## `draw-ellipse` *Generic function*

Summary      Draws an ellipse with the specified center and radius vectors.

Signature      `draw-ellipse drawable center-x center-y radius-1-dx radius-1-dy  
                   radius-2-dx radius-2-dy #key start-angle end-angle filled? => ()`  
`draw-ellipse* drawable center radius-1-dx radius-1-dy radius-2-dx  
                   radius-2-dy #key start-angle end-angle filled? => ()`

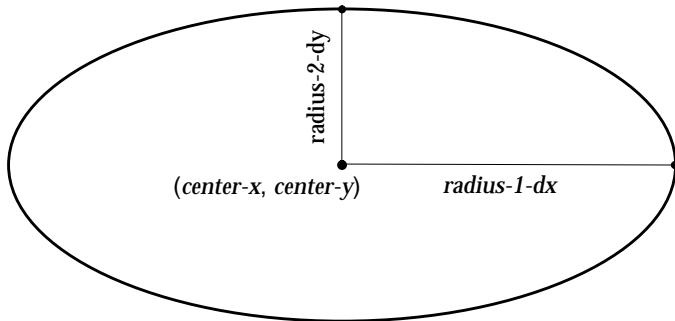
Arguments	<code>drawable</code>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	<code>radius-1-dx</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>radius-1-dy</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>radius-2-dx</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>radius-2-dy</code>	An instance of type <code>&lt;real&gt;</code> .
	<code>start-angle</code>	An instance of type <code>false-or(&lt;real&gt;)</code> .
	<code>end-angle</code>	An instance of type <code>false-or(&lt;real&gt;)</code> .
	<code>filled?</code>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .

The following arguments are specific to `draw-ellipse`.

<code>center-x</code>	An instance of type <code>&lt;real&gt;</code> .
<code>center-y</code>	An instance of type <code>&lt;real&gt;</code> .

The following argument is specific to `draw-ellipse*`.

<code>center</code>	An instance of type < <code>transform</code> >.
<code>Values</code>	None.
<code>Description</code>	<p>Draws an ellipse on <code>drawable</code> with the specified center and extreme points, using the current pen.</p> <p>The center of the ellipse is defined by (<code>center-x</code>, <code>center-y</code>), and the extreme points of the ellipse (that is, the points furthest away from the center for each radius) are calculated by adding the radius vectors <code>radius-1-dx</code> and <code>radius-1-dy</code> to <code>center-x</code> and <code>center-y</code> respectively (to calculate the outermost points for the first radius), and adding the radius vectors <code>radius-2-dx</code> and <code>radius-2-dy</code> to <code>center-x</code> and <code>center-y</code> respectively (to calculate the outermost points for the second radius).</p> <p>Please note that <code>draw-ellipse</code> does not currently support non-axis-aligned ellipses. For all practical purposes, this means that <code>radius-1-dy</code> and <code>radius-2-dx</code> must always be 0.</p>



The arguments `start-angle` and `end-angle` let you draw just a section of the ellipse, rather than the whole ellipse.

If `filled?` is `#t` then the ellipse will be filled, using the current brush.

The function `draw-ellipse*` is identical to `draw-ellipse`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also      `draw-circle`, page 366  
                 `draw-oval`, page 372

	<i>Generic function</i>
<b>draw-image</b>	
Summary	Draws the specified image at the specified position.
Signature	<code>draw-image drawable image x y =&gt; ()</code> <code>draw-image* drawable image point =&gt; ()</code>
Arguments	<p><i>drawable</i>      An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</p> <p><i>image</i>      An instance of type <code>&lt;image&gt;</code>.</p> <p>The following arguments are specific to <code>draw-image</code>.</p> <p><i>x</i>      An instance of type <code>&lt;real&gt;</code>.</p> <p><i>y</i>      An instance of type <code>&lt;real&gt;</code>.</p> <p>The following argument is specific to <code>draw-image*</code>.</p> <p><i>point</i>      An instance of type <code>&lt;transform&gt;</code>.</p>
Values	None.
Description	<p>Draws <i>image</i> on <i>drawable</i> at (<i>x,y</i>).</p> <p>The function <code>draw-image*</code> is identical to <code>draw-image</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>

See also      [draw-pixmap](#), page 373  
[draw-text](#), page 382

## **draw-line** *Generic function*

Summary      Draws a line between the specified points.

Signature     `draw-line drawable x1 y1 x2 y2 => ()`  
`draw-line* drawable point1 point2 => ()`

Arguments    `drawable`      An instance of type `type-union(<sheet>, <medium>)`.

The following arguments are specific to `draw-line`.

`x1`      An instance of type `<real>`.  
`y1`      An instance of type `<real>`.  
`x2`      An instance of type `<real>`.  
`y2`      An instance of type `<real>`.

The following arguments are specific to `draw-line*`.

`point1`      An instance of type `<transform>`.  
`point2`      An instance of type `<transform>`.

Values      None.

Description    Draws a line on `drawable` between  $(x_1, y_1)$  and  $(x_2, y_2)$ , using the current pen. Dashed lines start dashing from the first point.

The function `draw-line*` is identical to `draw-line`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also [curve-to](#), page 360  
[draw-arrow](#), page 363  
[draw-bezier-curve](#), page 364  
[draw-lines](#), page 371  
[draw-point](#), page 374  
[line-to](#), page 387

	<i>Generic function</i>
<b>draw-lines</b>	
Summary	Draws a series of lines between the specified sequence of points.
Signature	<code>draw-lines <i>drawable coord-seq</i> =&gt; ()</code> <code>draw-lines* <i>drawable points</i> =&gt; ()</code>
Arguments	<p><i>drawable</i>      An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</p> <p>The following argument is specific to <code>draw-lines</code>.</p> <p><i>coord-seq</i>      An instance of type <code>limited(&lt;sequence&gt;, of: &lt;coordinate&gt;)</code>.</p> <p>The following argument is specific to <code>draw-lines*</code>.</p> <p><i>points</i>      An instance of type <code>limited(&lt;sequence&gt;, of: &lt;point&gt;)</code>.</p>
Values	None.
Description	Draws a series of lines on <i>drawable</i> between the specified sequence of points, using the current pen. Dashed lines start dashing from the first point of each line.

The function `draw-lines*` is identical to `draw-line`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

**Example**

```
draw-lines(medium,
           vector(100, 150,
                  200, 250,
                  300, 350,
                  400, 450));
```

**See also**

`draw-line`, page 370  
`draw-points`, page 375  
`draw-rectangles`, page 379

**draw-oval***Generic function***Summary**

Draws an oval with the specified center and radii.

**Signature**

```
draw-oval  drawable center-x center-y x-radius y-radius #key filled?
=> ()
```

```
draw-oval*  drawable center x-radius y-radius #key filled? => ()
```

**Arguments**

**drawable** An instance of type `type-union(<sheet>, <medium>)`.

**x-radius** An instance of type `<real>`.

**y-radius** An instance of type `<real>`.

**filled?** An instance of type `<boolean>`. Default value: `#t`.

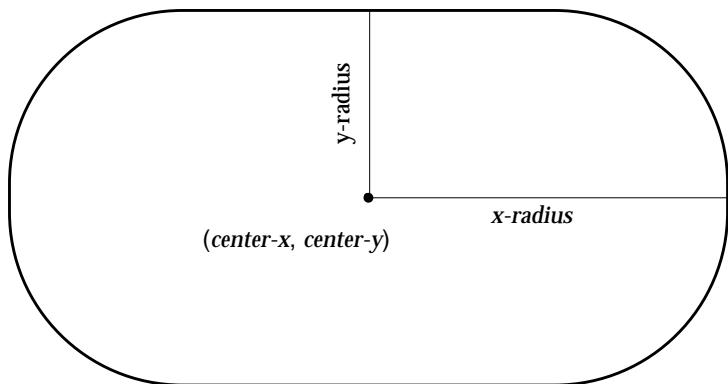
The following arguments are specific to `draw-oval`.

**center-x** An instance of type `<real>`.

**center-y** An instance of type `<real>`.

The following argument is specific to `draw-oval*`.

<i>center</i>	An instance of type <transform>.
Values	None.
Description	<p>Draws an oval on <i>drawable</i> with center (<i>center-x</i>,<i>center-y</i>) and radii defined by <i>x-radius</i> and <i>y-radius</i>, using the current pen.</p> <p>Ovals are similar to ellipses, except that they have straight edges.</p>



If *filled?* is #t then the oval will be filled, using the current brush.

The function **draw-oval\*** is identical to **draw-oval**, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also	<b>draw-circle</b> , page 366 <b>draw-ellipse</b> , page 367
----------	---

## draw-pixmap

*Generic function*

Summary	Draws the contents of the specified pixmap at the specified point.
---------	--

Signature	<code>draw-pixmap drawable pixmap x y #key function =&gt; ()</code> <code>draw-pixmap* drawable pixmap point #key function =&gt; ()</code>
Arguments	<p><i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</p> <p><i>pixmap</i> An instance of type <code>&lt;pixmap&gt;</code>.</p> <p><i>function</i> An instance of type <code>&lt;function&gt;</code>. Default value: <code>\$boolean-1</code>.</p>
	The following arguments are specific to <code>draw-pixmap</code> .
	<p><i>x</i> An instance of type <code>&lt;real&gt;</code>.</p> <p><i>y</i> An instance of type <code>&lt;real&gt;</code>.</p>
	The following argument is specific to <code>draw-pixmap*</code> .
	<i>point</i> An instance of type <code>&lt;transform&gt;</code> .
Values	None.
Description	<p>Draws the contents of <i>pixmap</i> on <i>drawable</i> at (x,y).</p> <p>The function <code>draw-pixmap*</code> is identical to <code>draw-pixmap</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>
See also	<p><code>destroy-pixmap</code>, page 361</p> <p><code>draw-image</code>, page 369</p> <p><code>draw-text</code>, page 382</p> <p><code>make-pixmap</code>, page 388</p>

**draw-point***Generic function*

Summary	Draws a single point at the specified position.
---------	---

Signature	<code>draw-point <i>drawable</i> <i>x</i> <i>y</i> =&gt; ()</code> <code>draw-point* <i>drawable</i> <i>point</i> =&gt; ()</code>
Arguments	<i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> . The following arguments are specific to <code>draw-point</code> . <i>x</i> The x coordinate. <i>y</i> The y coordinate.
	The following argument is specific to <code>draw-point*</code> . <i>point</i> An instance of type <code>&lt;transform&gt;</code> .
Values	None.
Description	Draws a single point on <i>drawable</i> at ( <i>x,y</i> ). The function <code>draw-point*</code> is identical to <code>draw-point</code> , except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.
See also	<code>draw-line</code> , page 370 <code>draw-points</code> , page 375

	<i>Generic function</i>
Summary	Draws a sequence of points at the specified positions.
Signature	<code>draw-points <i>drawable</i> <i>coord-seq</i> =&gt; ()</code> <code>draw-points* <i>drawable</i> <i>points</i> =&gt; ()</code>
Arguments	<i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> . The following argument is specific to <code>draw-points</code> .

<i>coord-seq</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;coordinate&gt;)</code> .
The following argument is specific to <code>draw-points*</code> .	
<i>points</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;point&gt;)</code> .
Values	None.
Description	<p>Draws a sequence of points on <i>drawable</i> at the specified positions.</p> <p>The function <code>draw-points*</code> is identical to <code>draw-points</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>
See also	<p><code>draw-lines</code>, page 371</p> <p><code>draw-point</code>, page 374</p> <p><code>draw-rectangles</code>, page 379</p>

## **draw-polygon** *Generic function*

Summary	Draws a polygon joining the specified points.						
Signature	<code>draw-polygon drawable coord-seq #key closed? filled? =&gt; ()</code> <code>draw-polygon* drawable points #key closed? filled? =&gt; ()</code>						
Arguments	<table> <tr> <td><i>drawable</i></td> <td>An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</td> </tr> <tr> <td><i>closed?</i></td> <td>An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</td> </tr> <tr> <td><i>filled?</i></td> <td>An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</td> </tr> </table>	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .	<i>closed?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .	<i>filled?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .						
<i>closed?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .						
<i>filled?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .						

The following argument is specific to `draw-polygon`.

*coord-seq* An instance of type `limited(<sequence>, of: <coordinate>)`.

The following argument is specific to `draw-polygon*`.

*points* An instance of type `limited(<sequence>, of: <point>)`.

Values None.

Description Draws a polygon on *drawable* joining the specified points, using the current pen. Dashed lines start dashing at the starting point of the first segment.

If *closed?* is `#t`, then the polygon is closed, that is, a line is drawn from the last point in the sequence back to the first.

If *filled?* is `#t` then the polygon will be filled, using the current brush.

The function `draw-polygon*` is identical to `draw-polygon`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also `draw-rectangle`, page 377

`draw-regular-polygon`, page 380

`draw-triangle`, page 384

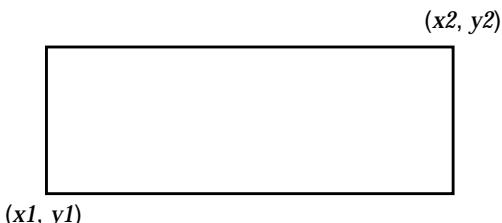
## `draw-rectangle`

## *Generic function*

Summary Draws a rectangle at the specified position.

Signature  
`draw-rectangle drawable x1 y1 x2 y2 #key filled? => ()`  
`draw-rectangle* drawable point1 point2 #key filled? => ()`

Arguments	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	<i>filled?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
The following arguments are specific to <code>draw-rectangle</code> .		
	<i>x1</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>y1</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>x2</i>	An instance of type <code>&lt;real&gt;</code> .
	<i>y2</i>	An instance of type <code>&lt;real&gt;</code> .
The following arguments are specific to <code>draw-rectangle*</code> .		
	<i>point1</i>	An instance of type <code>&lt;transform&gt;</code> .
	<i>point2</i>	An instance of type <code>&lt;transform&gt;</code> .
Values	None.	
Description	<p>Draws a rectangle on <i>drawable</i> with left and right corners at <math>(x_1, y_1)</math> and <math>(x_2, y_2)</math>, using the current pen. Dashed lines start dashing at the starting point of the first segment.</p> <p>Note that the specified points could represent either top or bottom corners: only one rectangle is possible between and pair of points.</p>	



If *filled?* is `#t` then the rectangle will be filled, using the current brush.

The function `draw-rectangle*` is identical to `draw-rectangle`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also `draw-polygon`, page 376  
`draw-rectangles`, page 379  
`draw-regular-polygon`, page 380  
`draw-triangle`, page 384

## `draw-rectangles` *Generic function*

Summary Draws a sequence of rectangles at the specified positions.

Signature `draw-rectangles` *drawable coord-seq #key filled? => ()*  
`draw-rectangles*` *drawable points #key filled? => ()*

Arguments *drawable* An instance of type `type-union(<sheet>, <medium>)`.

*filled?* An instance of type `<boolean>`. Default value: `#t`.

The following argument is specific to `draw-rectangles`.

*coord-seq* An instance of type `limited(<sequence>, of: <coordinate>)`.

The following argument is specific to `draw-rectangles*`.

*points* An instance of type `limited(<sequence>, of: <point>)`.

Values None.

Description	<p>Draws a sequence of rectangles on <i>drawable</i> with left and right corners at the specified positions, using the current pen. Dashed lines start dashing at the starting point of the first segment of each rectangle.</p> <p>If <i>filled?</i> is #t then the rectangles will be filled, using the current brush.</p> <p>The function <code>draw-rectangles*</code> is identical to <code>draw-rectangles</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>
See also	<p><code>draw-lines</code>, page 371</p> <p><code>draw-points</code>, page 375</p> <p><code>draw-rectangle</code>, page 377</p>

	<i>Generic function</i>								
Summary	Draws a regular polygon that touches the specified points, and has the specified number of sides.								
Signature	<pre><code>draw-regular-polygon <i>drawable</i> <i>x1</i> <i>y1</i> <i>x2</i> <i>y2</i> <i>nsides</i> #key handedness closed? filled? =&gt; ()</code></pre> <pre><code>draw-regular-polygon* <i>drawable</i> <i>point1</i> <i>point2</i> <i>nsides</i> #key handedness closed? filled? =&gt; ()</code></pre>								
Arguments	<table> <tr> <td><i>drawable</i></td><td>An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</td></tr> <tr> <td><i>nsides</i></td><td>An instance of type <code>&lt;integer&gt;</code>.</td></tr> <tr> <td><i>handedness</i></td><td>Default value: #"<code>left</code>".</td></tr> <tr> <td><i>closed?</i></td><td>An instance of type <code>&lt;boolean&gt;</code>. Default value: #t.</td></tr> </table>	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .	<i>nsides</i>	An instance of type <code>&lt;integer&gt;</code> .	<i>handedness</i>	Default value: #" <code>left</code> ".	<i>closed?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: #t.
<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .								
<i>nsides</i>	An instance of type <code>&lt;integer&gt;</code> .								
<i>handedness</i>	Default value: #" <code>left</code> ".								
<i>closed?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: #t.								

*filled?* An instance of type `<boolean>`. Default value: `#t`.

The following arguments are specific to `draw-regular-polygon`.

*x1* An instance of type `<real>`.

*y1* An instance of type `<real>`.

*x2* An instance of type `<real>`.

*y2* An instance of type `<real>`.

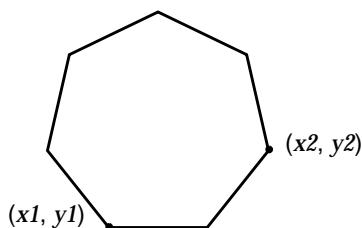
The following arguments are specific to `draw-regular-polygon*`.

*point1* An instance of type `<transform>`.

*point2* An instance of type `<transform>`.

Values None.

Description Draws a regular polygon on *drawable*, using the current pen, that touches the specified points, and has the specified number of sides. Dashed lines start dashing at the starting point of the first segment.



If *filled?* is `#t` then the polygon will be filled, using the current brush.

The function `draw-regular-polygon*` is identical to `draw-regular-polygon`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also      `draw-polygon`, page 376  
                 `draw-rectangle`, page 377  
                 `draw-triangle`, page 384

<b>draw-text</b>	<i>Generic function</i>												
Summary	Draws text at the specified point, in a specified direction.												
Signature	<code>draw-text</code> <i>drawable text x y #key start end align-x align-y towards-point transform-glyphs? =&gt; ()</i> <code>draw-text*</code> <i>drawable text point #key start end align-x align-y towards-point transform-glyphs? =&gt; ()</i>												
Arguments	<table border="0"> <tr> <td><i>drawable</i></td><td>An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code>.</td></tr> <tr> <td><i>text</i></td><td>An instance of type <code>type-union(&lt;string&gt;, &lt;character&gt;)</code>.</td></tr> <tr> <td><i>start</i></td><td>An instance of type <code>&lt;integer&gt;</code>. Default value: 0.</td></tr> <tr> <td><i>end</i></td><td>An instance of type <code>&lt;integer&gt;</code>. Default value: <code>size(text)</code>.</td></tr> <tr> <td><i>align-x</i></td><td>An instance of type <code>one-of(#"left", #"right", #"center")</code>. Default value: <code>"left"</code>.</td></tr> <tr> <td><i>align-y</i></td><td>An instance of type <code>one-of(#"top", #"bottom", #"baseline")</code>. Default value: <code>"baseline"</code>.</td></tr> </table>	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .	<i>text</i>	An instance of type <code>type-union(&lt;string&gt;, &lt;character&gt;)</code> .	<i>start</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: 0.	<i>end</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <code>size(text)</code> .	<i>align-x</i>	An instance of type <code>one-of(#"left", #"right", #"center")</code> . Default value: <code>"left"</code> .	<i>align-y</i>	An instance of type <code>one-of(#"top", #"bottom", #"baseline")</code> . Default value: <code>"baseline"</code> .
<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .												
<i>text</i>	An instance of type <code>type-union(&lt;string&gt;, &lt;character&gt;)</code> .												
<i>start</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: 0.												
<i>end</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <code>size(text)</code> .												
<i>align-x</i>	An instance of type <code>one-of(#"left", #"right", #"center")</code> . Default value: <code>"left"</code> .												
<i>align-y</i>	An instance of type <code>one-of(#"top", #"bottom", #"baseline")</code> . Default value: <code>"baseline"</code> .												

*transform-glyphs?* An instance of type `<boolean>`. Default value: `#f`.

*do-tabs?* An instance of type `<boolean>`. Default value: `#f`

The following arguments are specific to `draw-text`.

*towards-x* An instance of type `<real>`.

*towards-y* An instance of type `<real>`.

*x* An instance of type `<real>`.

*y* An instance of type `<real>`.

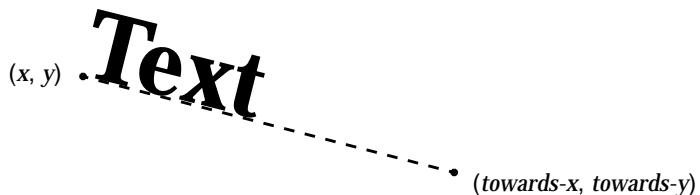
The following arguments are specific to `draw-text*`.

*towards-point* An instance of type `<transform>`.

*point* An instance of type `<transform>`.

Values None.

Description Draws text from *text* on *drawable* at  $(x,y)$ . Text is drawn in the direction of the point  $(\text{towards-}x, \text{towards-}y)$ .



If *start* and *end* are specified, then only a section of text is drawn, starting at character *start*, and ending with character *end*. By default, the whole of *text* is drawn.

The *align-x* and *align-y* arguments let you specify the left-right alignment and the top-bottom alignment (respectively) of the text that is written to *drawable*.

For *align-x*, the whole of the distance between  $(x,y)$  and  $(\text{towards}-x,\text{towards}-y)$  is used to align *text*. Thus, if *align-x* is `#"right"`, the text will appear closer to  $(\text{towards}-x,\text{towards}-y)$  than to  $(x,y)$ , assuming *text* occupies less space than the distance between these two points.

The argument *transform-glyphs?* controls whether the text is reversed in cases when *towards-x* is less than *x*. If *transform-glyphs?* is `#t`, then text is reversed in these cases, that is, the last character of *text* to be written is still closest to the point  $(\text{towards}-x,\text{towards}-y)$ , and the text appears reversed. If *transform-glyphs?* is `#f`, then the first character of *text* to be written is closest to the point  $(\text{towards}-x,\text{towards}-y)$ , and the text does not appear reversed.

If *do-tabs?* is `#t`, then any tab characters in *text* are honored, and are drawn as tabs. If *do-tabs?* is `#f`, then tab characters are replaced by spaces.

The function `draw-text*` is identical to `draw-text`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also      [draw-image](#), page 369  
[draw-pixmap](#), page 373

## **draw-triangle** *Generic function*

Summary	Draws a triangle between the specified points.
Signature	<code>draw-triangle <i>drawable</i> x1 y1 x2 y2 x3 y3 #key filled? =&gt; ()</code> <code>draw-triangle* <i>drawable</i> p1 p2 p3 #key filled? =&gt; ()</code>
Arguments	<b><i>drawable</i></b> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .

*filled?* An instance of type `<boolean>`. Default value: `#t`.

The following arguments are specific to `draw-triangle`.

*x1* An instance of type `<real>`.

*y1* An instance of type `<real>`.

*x2* An instance of type `<real>`.

*y2* An instance of type `<real>`.

*x3* An instance of type `<real>`.

*y3* An instance of type `<real>`.

The following arguments are specific to `draw-triangle*`.

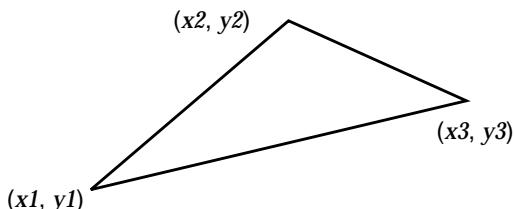
*p1* An instance of type `<transform>`.

*p2* An instance of type `<transform>`.

*p3* An instance of type `<transform>`.

Values None.

Description Draws a triangle on *drawable* between the specified points, using the current pen. Dashed lines start dashing at the starting point of the first segment.



If *filled?* is `#t` then the triangle will be filled, using the current brush.

The function `draw-triangle*` is identical to `draw-triangle`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also      `draw-polygon`, page 376  
                `draw-rectangle`, page 377  
                `draw-regular-polygon`, page 380

<b>end-path</b>	<i>Generic function</i>
Summary	Ends the definition of the current path in the specified drawable object.
Signature	<code>end-path</code> <i>drawable</i> => ()
Arguments	<i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
Values	None.
Description	Ends the definition of the current path in <i>drawable</i> . Once the definition has been ended, the path can be rendered to the drawable using <code>fill-path</code> or <code>stroke-path</code> .  The function <code>close-path</code> can also be used to end the definition of a path.
See also	<code>abort-path</code> , page 354 <code>close-path</code> , page 356 <code>start-path</code> , page 392

## **fill-path** *Generic function*

Summary	Uses the current brush to fill the current path on the specified drawable object.
Signature	<b>fill-path</b> <i>drawable</i> => ()
Arguments	<i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
Values	None.
Description	Uses the current brush to fill the current path on <i>drawable</i> . If the path has not already been closed using <code>close-path</code> , it is closed automatically.
See also	<code>stroke-path</code> , page 393 <code>close-path</code> , page 356

## **line-to** *Generic function*

Summary	Draws a line from the current position in the path to a new position.
Signature	<b>line-to</b> <i>drawable</i> <i>x</i> <i>y</i> => () <b>line-to*</b> <i>drawable</i> <i>point</i> => ()
Arguments	<i>drawable</i> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
	The following arguments are specific to <code>line-to</code> . <i>x</i> An instance of type <code>&lt;real&gt;</code> . <i>y</i> An instance of type <code>&lt;real&gt;</code> .
	The following argument is specific to <code>line-to*</code> .

	<i>point</i>	An instance of type < <b>transform</b> >.
Values	None.	
Description	<p>Draws a line from the current position in the path to (x,y).</p> <p>This function is used, in combination with <code>move-to</code>, <code>curve-to</code>, and <code>arc-to</code>, to define a path. The function <code>start-path</code> should be used to start the definition of the path, and <code>end-path</code> can be used to finish the definition.</p> <p>The function <code>line-to*</code> is identical to <code>line-to</code>, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.</p>	
See also	<p><code>arc-to</code>, page 354</p> <p><code>curve-to</code>, page 360</p> <p><code>draw-bezier-curve</code>, page 364</p> <p><code>draw-line</code>, page 370</p> <p><code>move-to</code>, page 389</p>	

## make-pixmap *Generic function*

Summary	Creates a pixmap from the specified medium with a specified size.	
Signature	<code>make-pixmap medium width height =&gt; pixmap</code>	
Arguments	<code>medium</code>	An instance of type < <b>medium</b> >.
	<code>width</code>	An instance of type < <b>integer</b> >.
	<code>height</code>	An instance of type < <b>integer</b> >.
Values	<code>pixmap</code>	An instance of type < <b>pixmap</b> >.

Description	Creates a pixmap from <i>medium</i> with a specified size, in pixels, given by <i>width</i> and <i>height</i> .
See also	<a href="#">draw-pixmap</a> , page 373 <a href="#">&lt;pixmap&gt;</a> , page 390 <a href="#">pixmap?</a> , page 391

## move-to *Generic function*

Summary	Move the position in the current path on the specified drawable.
Signature	<code>move-to <b>drawable</b> <i>x</i> <i>y</i> =&gt; ()</code> <code>move-to* <b>drawable</b> <i>point</i> =&gt; ()</code>
Arguments	<b>drawable</b> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> . The following arguments are specific to <code>move-to</code> . <b>x</b> An instance of type <code>&lt;real&gt;</code> . <b>y</b> An instance of type <code>&lt;real&gt;</code> . The following argument is specific to <code>move-to*</code> . <b>point</b> An instance of type <code>&lt;transform&gt;</code> .
Values	None.
Description	Move the position in the current path on <i>drawable</i> to ( <i>x,y</i> ). This function is used, in combination with <code>line-to</code> , <code>curve-to</code> , and <code>arc-to</code> , to define a path. The function <code>start-path</code> should be used to start the definition of the path, and <code>end-path</code> can be used to finish the definition.

The function `move-to` can be used several times within the definition of a path, allowing for the definition of several visually separate drawings within the same path.

The function `move-to*` is identical to `move-to`, except that it passes composite objects, rather than separate coordinates, in its arguments. You should be aware that using this function may lead to a loss of performance.

See also      `arc-to`, page 354  
                `curve-to`, page 360  
                `line-to`, page 387

## **<pixmap>** *Open abstract instantiable class*

Summary      The class of pixmap objects.

Superclasses      `<image>`

Init-keywords      None.

Description      The class of pixmap objects.

A pixmap can be thought of as an “off-screen window”, that is, a medium that can be used for graphical output, but is not visible on any display device. Pixmaps are provided to allow you to generate a piece of output associated with some display device that can then be rapidly drawn on a real display device. For example, an electrical CAD system might generate a pixmap that corresponds to a complex, frequently used part in a VLSI schematic, and then use `copy-from-pixmap` to draw the part as needed.

Operations      The following operation is exported from the `DUIM-Graphics` module.

```
copy-from-pixmap destroy-pixmap draw-image
draw-pixmap  pixmap?
```

The following operation is exported from the **DUIL-DCS** module.

```
image-height image-width
```

See also **draw-pixmap**, page 373  
**make-pixmap**, page 388  
**pixmap?**, page 391

## **pixmap?** *Generic function*

Summary Returns true if the specified object is a pixmap.  
Signature **pixmap? object => pixmap?**  
Arguments **object** An instance of type **<object>**.  
Values **pixmap?** An instance of type **<boolean>**.  
Description Returns true if *object* is a pixmap.  
See also **<pixmap>**, page 390

## **<pixmap-medium>** *Open abstract instantiable class*

Summary The class of pixmap mediums.  
Superclasses **<medium>**  
Init-keywords None.

Description	The class of pixmap mediums, that is mediums capable of doing output to a pixmap.
Operations	<code>with-output-to-pixmap</code>
See also	<a href="#">`&lt;medium&gt;`</a> , page 259 <a href="#">`with-output-to-pixmap`</a> , page 393

## restore-clipping-region

	<i>Generic function</i>
<b>start-path</b>	
Summary	Starts a new path on the specified drawable object.
Signature	<code>start-path <i>drawable</i> =&gt; ()</code>
Arguments	<b><i>drawable</i></b> An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
Values	None.
Description	<p>Starts a new path on <i>drawable</i>. The path can be created with any number of calls to <code>line-to</code>, <code>curve-to</code> and <code>move-to</code>. Its appearance can also be manipulated using <code>fill-path</code> and <code>stroke-path</code>.</p> <p>After creating the path, use either <code>close-path</code> or <code>end-path</code> to finish the path, or <code>abort-path</code> to abandon it altogether.</p>
See also	<a href="#">`abort-path`</a> , page 354 <a href="#">`close-path`</a> , page 356 <a href="#">`end-path`</a> , page 386

**stroke-path***Generic function*

Summary	Uses the current pen to draw the current path on the specified drawable object.	
Signature	<code>stroke-path drawable =&gt; ()</code>	
Arguments	<i>drawable</i>	An instance of type <code>type-union(&lt;sheet&gt;, &lt;medium&gt;)</code> .
Values	None.	
Description	<p>Uses the current pen to draw the current path on <i>drawable</i>. Note that the path must not have been previously filled. This function does not close the path: you must use <code>close-path</code> if you wish to do this.</p>	
See also	<a href="#">close-path</a> , page 356 <a href="#">fill-path</a> , page 387	

**with-output-to-pixmap***Macro*

Summary	Executes a body of code, returning the results to a pixmap.	
Macro call	<code>with-output-to-pixmap (medium, #rest options) body end =&gt; pixmap</code>	
Arguments	<i>medium</i>	An instance of type <code>&lt;pixmap-medium&gt;</code> .
	<i>options</i>	An instance of type <code>&lt;object&gt;</code> .
	<i>body</i>	An instance of type <code>&lt;object&gt;</code> .
Values	<i>pixmap</i>	An instance of type <code>&lt;pixmap&gt;</code> .

Description	<p>Executes a body of code, returning the results to a pixmap. Binds <i>medium</i> to a pixmap medium, that is, a medium that does output to a pixmap, and then evaluates <i>body</i> in that context. All the output done to <i>medium</i> inside of <i>body</i> is drawn on the pixmap stream. The pixmap medium supports the medium output protocol, including all of the graphics functions.</p> <p>The returned value is a pixmap that can be drawn onto <i>medium</i> using <code>copy-from-pixmap</code>.</p>
See also	<p><code>do-with-output-to-pixmap</code>, page 362</p> <p><code>&lt;pixmap-medium&gt;</code>, page 391</p>

# 7

---

---

# DUIM-Layouts Library

## 7.1 Overview

The DUIM-Layouts library contains interfaces that define a number of layouts for use in your GUI applications, as well as the necessary functions, generic functions, and macros for creating, manipulating, and calculating them automatically. The library contains a single module, `duim-layouts`, from which all the interfaces described in this chapter are exposed. Section 7.3 on page 399 contains complete reference entries for each exposed interface.

Layouts are sheet objects that determine how the interface elements are presented on the screen. A layout object takes a number of children, expressed as a vector, and lays out those children according to certain constraints. Each child of a layout must be an instance of a DUIM class.

## 7.2 The class hierarchy for DUIM-Layouts

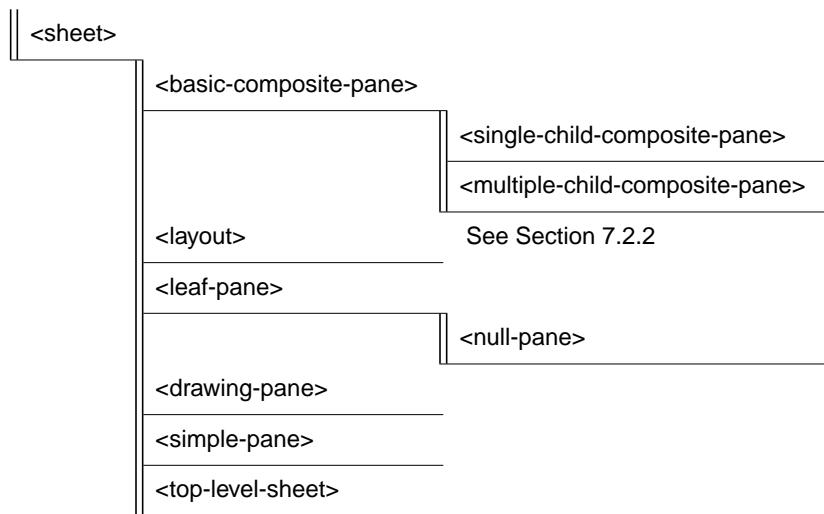
This section presents an overview of the available classes of layout, and describes the class hierarchy present.

### 7.2.1 The `<layout>` class and its subclasses

The base class for the majority of DUIM layouts is the `<layout>` class, which is itself a subclass of `<sheet>`. All other layout-oriented classes are subclasses of `<sheet>`.

The immediate subclasses of `<sheet>` that are exposed by the DUIM-Layouts library are shown in Table 7.1. Only `<basic-composite-pane>`, `<leaf-pane>`, and `<layout>` have any subclasses defined. See Section 7.2.2 on page 397 for details of the subclasses of `<layout>`.

**Table 7.1** Overall class hierarchy for the DUIM-Layouts library



All the actual layouts provided by the DUIM-Layouts library are subclasses of the base `<layout>` class, and are described in Section 7.2.2. In addition, a number of different types of pane are supplied by the DUIM-Layouts library.

#### `<basic-composite-pane>`

This is a basic type of pane that is used to create any sheet that can contain children. It has two subclasses, one used for sheets that take only a single child, and one for sheets that can take several children.

**<drawing-pane>**

This type of pane is used to create sheets on which geometric objects are drawn, for example, using the function provided by the DUIM-Geometry module or the DUIM-Graphics module. For more information on these modules, see Chapter 2, “DUIM-Geometry Library”, and Chapter 6, “DUIM-Graphics Library”, respectively.

**<top-level-sheet>**

This class is used for any sheets that are at the top level of the hierarchy of windows on the screen: that is, there is no other sheet that is the parent of an instance of **<top-level-sheet>**.

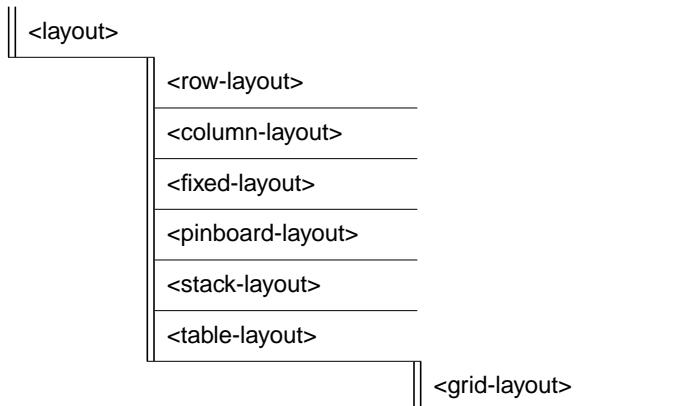
**<leaf-pane>** In contrast to **<top-level-sheet>**, an instance of **<leaf-pane>** cannot have any children, and is at the end of the hierarchy of windows on the screen.

**<simple-pane>** This class is the most basic type of pane, and is used when no other more suitable class is available.

## 7.2.2 Subclasses of <layout>

The subclasses of **<layout>** are shown in Table 7.2.

**Table 7.2** Subclasses of the **<layout>** class



The layouts provided by DUIM fall roughly into two categories:

- Layout classes that calculate the position and size of their children for you, subject to some constraints.
- Layout classes that let you specify precisely the position of their children, and, optionally, the size of the children as well.

The classes of layout available are as follows:

**<column-layout>**

This class lays out its children in a single column, with all its children left-aligned by default.

**<row-layout>** This class lays out its children in a single row.

**<stack-layout>** This class lays out its children one on top of another, aligned at the top left corner by default. It is specifically for windows that contain a number of layouts, only one of which is visible at any one time, such as property sheets, tab controls, or wizards.

**<table-layout>** This class lays out its children in a table, according to a specified number of rows and columns.

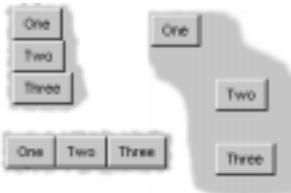
**<pinboard-layout>**

This does not constrain the position of its children in any way. It is up to you to position each child individually, like pins on a pinboard.

**<fixed-layout>** This class is like **<pinboard-layout>**, in that you must specify the position of each child. Unlike **<pinboard-layout>**, however, you must also specify the size of each child.

In addition to the basic types of layout described above, a subclass of **<table-layout>** is provided, as follows:

**<grid-layout>** This is a specialized version of **<table-layout>**, in which all the cells in the table are forced to be the same size.



**Figure 7.1** Column, row, and pinboard layouts

## 7.3 DUIM-Layouts Module

This section contains a complete reference of all the interfaces that are exported from the `duim-layouts` module.

	<i>Open generic function</i>						
<b>allocate-space</b>							
Summary	Allocates space within a layout for its children.						
Signature	<code>allocate-space pane width height =&gt; ()</code>						
Arguments	<table border="0"> <tr> <td><i>pane</i></td><td>An instance of type <code>&lt;sheet&gt;</code>.</td></tr> <tr> <td><i>width</i></td><td>An instance of type <code>&lt;integer&gt;</code>.</td></tr> <tr> <td><i>height</i></td><td>An instance of type <code>&lt;integer&gt;</code>.</td></tr> </table>	<i>pane</i>	An instance of type <code>&lt;sheet&gt;</code> .	<i>width</i>	An instance of type <code>&lt;integer&gt;</code> .	<i>height</i>	An instance of type <code>&lt;integer&gt;</code> .
<i>pane</i>	An instance of type <code>&lt;sheet&gt;</code> .						
<i>width</i>	An instance of type <code>&lt;integer&gt;</code> .						
<i>height</i>	An instance of type <code>&lt;integer&gt;</code> .						
Values	None.						
Description	Allocates space within a layout for its children. During the space allocation pass, a composite pane arranges its children within the available space and allocates space to them according to their space requirements and its own composition rules by calling <code>allocate-space</code> on each of the child panes. For example, <code>&lt;column-layout&gt;</code> arranges all its chil-						

dren in a vertical column. The `width` and `height` arguments are the width and height of `pane` in device units, that is, pixels. These arguments give the amount of space into which all children must fit.

This function actually calls `do-allocate-space` to perform the calculations. Client code may specialize `do-allocate-space`, but not call it. Call `allocate-space` instead.

See also      [do-allocate-space, page 407](#)

## <basic-user-pane> *Base class*

Summary      The class of basic user panes.

Superclasses      <wrapping-layout-pane>

Init-keywords      **`region:`**      An instance of type <`region`>. Default value: `$nowhere`.

**`transform:`**      An instance of type <`transform`>. Default value: `$identity-transform`.

**`port:`**      An instance of type `false-or(<port>)`. Default value: `#f`.

**`style-descriptor:`**      An instance of type `false-or(<object>)`. Default value: `#f`.

**`help-context:`**      An instance of type <`object-table`>. Default value: `make(<object-table>)`.

**`help-source:`**      An instance of type <`object-table`>. Default value: `make(<object-table>)`.

Description      The class of basic user panes. This is the class that gets subclassed by `define-pane`.

You specify where on the screen the pane is to be displayed using the `region:` init-keyword. The region specified should be relative to the top left corner of the pane's parent, since the pane must be displayed within the confines of its parent.

If you wish the location of the pane to be transformed in some way, use the `transform:` init-keyword.

If you wish to use a port other than the default port, use the `port:` init-keyword.

You can specify the appearance for text in the pane using the `style-descriptor:` init-keyword.

The `help-source:` and `help-context:` keywords let you specify pointers to valid information available in any online help you supply with your application. The `help-context:` keyword should specify a context-ID present in the online help. This context-ID identifies the help topic that is applicable to the current pane. The `help-source:` init-keyword identifies the source file in which the help topic identified by `help-context:` can be found. A list of context-IDs should be provided by the author of the online help system.

Operations      None.

See also      `define pane`, page 406

## <column-layout>

*Open abstract instantiable class*

Summary      The class of column layouts.

Superclasses      <layout>

Init-keywords      `border:`      An instance of type <integer>. Default value: 0.

```

spacing:, y-spacing:
    An instance of type <integer>. Default
    value: 0.

equalize-heights?:
    An instance of type <boolean>. Default
    value: #f.

equalize-widths?:
    An instance of type <boolean>. Default
    value: #f.

x-alignment: An instance of type one-of(#"left",
    #"right", #"center"). Default value:
    #"left".

ratios:, y-ratios:
    An instance of type false-
    or(limited(<sequence>), of:
    <integer>)). Default value: #f.

```

Description	The class of column layouts. A column layout arranges its children in a column, automatically calculating the size and placement of each child within the specified parameters.
-------------	---



**Figure 7.2** Three buttons arranged in a column layout

The **border:** init-keyword provides a border of whitespace around the children in the layout, and the value of this init-keyword represents the size of the border in pixels. This basically has the same effect as using the macro **with-spacing** around the layout, except it uses a simpler syntax.

The **spacing:** or **y-spacing:** init-keywords let you specify how much vertical space should be inserted, in pixels, between the children of the layout. These two init-keywords can be used interchangeably.

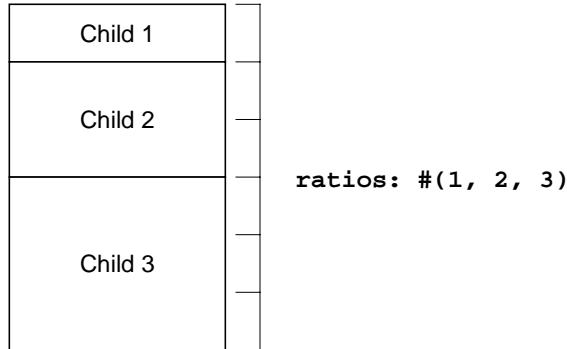
If true, **equalize-heights?:** ensures that all the children of the layout have the same height.

If true, **equalize-widths?:** ensures that all the children of the layout have the same width.

By default, all the children of a column layout are left-aligned. You can specify that they should be right or center-aligned using the **x-alignment:** keyword.

The **ratios:** or **y-ratios:** init-keywords let you specify the proportion of the total layout that should be taken up by each individual child. These two init-keywords can be used interchangeably.

The value passed to **ratios:** needs to be a sequence of as many integers as there are children in the layout. Each child is then allocated the appropriate portion of vertical space in the layout. For example, if the value #(1, 2, 3) is specified for the **ratios:** init-keyword of a column layout containing three children, then the first child would claim a sixth of the available vertical space, the second child would claim a third of the vertical space, and the third child would claim half the vertical space, as shown in the diagram below.



Operations      None.

Example      

```
contain(make(<column-layout>,
            children: vector(make(<button>,
                                  label: "Hello"),
                            make(<button>,
                                  label: "World")),
            spacing: 100,
            x-alignment: #"right",
            ratios: #(1, 3)));
```

See also      [`<grid-layout>`](#), page 411  
[`<layout>`](#), page 413  
[`<row-layout>`](#), page 423  
[`<stack-layout>`](#), page 433  
[`<table-layout>`](#), page 436  
[`vertically`](#), page 442

## **compose-space** *Generic function*

Summary      Returns the amount of space required for a specified child of a composite pane.

Signature	<code>compose-space pane #key width height =&gt; space-req</code>	
Arguments	<code>pane</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>width</code>	An instance of type <code>&lt;integer&gt;</code> .
	<code>height</code>	An instance of type <code>&lt;integer&gt;</code> .
Values	<code>space-req</code>	An instance of type <code>&lt;space-requirement&gt;</code> .
Description	<p>Returns the amount of space required for <code>pane</code>, which is a child of a composite pane. During the space composition pass, a composite pane will typically ask each of its children how much space it requires by calling <code>compose-space</code>. They answer by returning instances of <code>&lt;space-requirement&gt;</code>. The composite pane then forms its own space requirement by composing the space requirements of its children according to its own rules for laying out its children.</p> <p>The value returned by <code>compose-space</code> is an instance of <code>&lt;space-requirement&gt;</code> that represents how much space <code>pane</code> requires.</p> <p>The <code>width</code> and <code>height</code> arguments are real numbers that the <code>compose-space</code> method for a pane may use as “recommended” values for the width and height of the pane. These are used to drive top-down layout.</p> <p>This function actually calls <code>do-compose-space</code> to perform the space calculations. Client code may specialize <code>do-compose-space</code> but should not call it. Call <code>compose-space</code> instead.</p>	
See also	<p><code>do-compose-space</code>, page 408</p> <p><code>&lt;space-requirement&gt;</code>, page 427</p>	

<b>current-pane</b>	<i>Generic function</i>
Summary	Returns the current pane.

Signature	<code>current-pane =&gt; pane</code>	
Arguments	None.	
Values	<i>pane</i>	An instance of type <code>&lt;sheet&gt;</code> .
Description	Returns the current pane: that is, the pane that has the mouse focus.	

## define pane *Definition macro*

Summary	Defines a new class of DUIM pane.						
Macro call	<code>define pane name ({supers},*) {slots-and-panes} end</code>						
Arguments	<table> <tr> <td><i>name</i></td> <td>A Dylan name<sub>bnf</sub>.</td> </tr> <tr> <td><i>supers</i></td> <td>A Dylan name<sub>bnf</sub>.</td> </tr> <tr> <td><i>slots-and-panes</i></td> <td>A Dylan body<sub>bnf</sub>.</td> </tr> </table>	<i>name</i>	A Dylan name <sub>bnf</sub> .	<i>supers</i>	A Dylan name <sub>bnf</sub> .	<i>slots-and-panes</i>	A Dylan body <sub>bnf</sub> .
<i>name</i>	A Dylan name <sub>bnf</sub> .						
<i>supers</i>	A Dylan name <sub>bnf</sub> .						
<i>slots-and-panes</i>	A Dylan body <sub>bnf</sub> .						
Values	None.						
Description	<p>This macro lets you define a new class of DUIM pane. The <i>name</i> argument represents the name of the new class of pane, and <i>supers</i> is a list of zero or more superclasses for the new class. Multiple superclass names are separated by commas.</p> <p>The <i>slots-and-panes</i> argument represents the slot information for the new class, together with any init-keywords and default values that the slots should take.</p> <p>Panes are sheets which represent a “useful unit” in a GUI. There is no protocol class called <code>&lt;pane&gt;</code>.</p>						

- In most cases (such as when defining a frame using `define frame`), a pane class groups existing gadgets (or panes) to form effectively a new gadget, without actually creating a new class of `<gadget>`.
- Sometimes, a pane class implements some complex output-only sheet.
- Sometimes, a pane class implements the `<sheet>` part of a `<gadget>`.

In general, a pane is best described as a *concrete sheet*.

Example

```
define pane <my-pane> ()
  slot my-layout,
    init-keyword: layout:;;
  slot my-exit-buttons,
    init-keyword: exit-buttons:;;
end pane <my-pane>;
```

See also

`define frame`, page 650

## do-allocate-space

*Open generic function*

Summary

Called by `allocate-space` to calculate space requirements for a pane.

Signature

`do-allocate-space pane width height => ()`

Arguments

<code>pane</code>	An instance of type <code>&lt;sheet&gt;</code> .
<code>width</code>	An instance of type <code>&lt;integer&gt;</code> .
<code>height</code>	An instance of type <code>&lt;integer&gt;</code> .

Values

None.

Description	This function is called by <code>allocate-space</code> to calculate space requirements for a pane. When calculating space requirements for classes of pane you have defined yourself, you should add methods to this function, but not call it directly. Call <code>allocate-space</code> instead.
See also	<code>allocate-space</code> , page 399

## do-compose-space *Open generic function*

Summary	Called by <code>compose-space</code> to calculate space requirements for a child.	
Signature	<code>do-compose-space pane #key width height =&gt; space-req</code>	
Arguments	<code>pane</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>width</code>	An instance of type <code>&lt;integer&gt;</code> .
	<code>height</code>	An instance of type <code>&lt;integer&gt;</code> .
Values	<code>space-req</code>	An instance of type <code>&lt;space-requirement&gt;</code> .
Description	This function is called by <code>compose-space</code> to calculate space requirements for a child. When calculating space requirements for children in classes of pane you have defined yourself, you should specialize this function by adding methods for it. However, you should not call <code>do-compose-space</code> explicitly: call <code>compose-space</code> instead.	

Example	Assume that you have defined a new class of scroll bar as follows:
<pre>define class &lt;my-scroll-bar&gt;   (&lt;scroll-bar&gt;,    &lt;leaf-pane&gt;) end class &lt;test-scrollbar&gt;;</pre>	

A new method for do-compose-space can be defined as follows:

```
define method do-compose-space
  (pane :: <my-scroll-bar>, #key width, height)
  => (space-req :: <space-requirement>)
    select (gadget-orientation(pane))
      #"horizontal" =>
        make(<space-requirement>,
          width: width | 50,
          min-width: 50,
          max-width: $fill,
          height: 10);
      #"vertical" =>
        make(<space-requirement>,
          width: 10,
          height: height | 50,
          min-height: 50,
          max-height: $fill);
    end
end method do-compose-space;
```

See also      [compose-space](#), page 404

## <drawing-pane>

*Open abstract instantiable class*

Summary      The class of drawing panes.

Superclasses      <layout>

Init-keywords      **display-function:**

An instance of type **false-or(<function>)**.  
Default value: #f.

Description      The class of drawing panes. This is a pane that provides event handling and a drawing surface. Note that a drawing pane can be wrapped around a layout pane to provide a medium for all the children of the layout pane.

The `display-function:` init-keyword defines the display function for the pane. This gets called by the `handle-repaint` method for `<simple-pane>`.

Operations	None.
See also	<a href="#">handle-repaint, page 253</a> <a href="#">pane-display-function, page 420</a> <a href="#">&lt;simple-pane&gt;, page 426</a>

## \$fill *Constant*

Summary	Default value for width and height init-keywords for layout panes.
Type	<code>&lt;integer&gt;</code>
Value	100000
Description	<p>This constant is used as the default value for any <code>width:</code> and <code>height:</code> init-keywords in layout panes.</p> <p>These defaults gives the intuitive behavior that specifying only the width or height of a pane causes it to be allocated at least that much space, and it may be given extra space if there is extra space in the layout. This default behavior can be changed if either the <code>min-width:</code> or <code>min-height:</code> init-keywords are specified explicitly.</p>
See also	<a href="#">make, page 418</a>

## <fixed-layout> *Open abstract instantiable class*

Summary	The class of fixed layouts.
---------	-----------------------------

Superclasses	<code>&lt;layout&gt;</code>
Init-keywords	None.
Description	<p>The class of fixed layouts. Fixed layouts are similar to pinboard layouts, in that the positioning and geometry of the children of a fixed layout are entirely determined by the programmer. You can place children at any point in a fixed layout, and the layout does not attempt to calculate an optimum position or size for any of them.</p> <p>Fixed layouts differ from pinboard layouts, however, in that any children placed in a fixed layout are left at exactly the size and position that they were created: pinboard layouts leave the positions of any children alone, but constrains the sizes of the children to obey any constraints they have been given.</p> <p>Fixed layouts are most useful if you know exactly what size and position every child in the layout should be.</p>
Operations	None.
See also	<p><code>&lt;layout&gt;</code>, page 413</p> <p><code>&lt;pinboard-layout&gt;</code>, page 421</p>

<b><code>&lt;grid-layout&gt;</code></b>		<i>Open abstract instantiable class</i>
Summary	The class of grid layouts.	
Superclasses	<code>&lt;table-layout&gt;</code>	
Init-keywords	<code>cell-space-requirement:</code>	An instance of type <code>&lt;space-requirement&gt;</code> .

Description	The class of grid layouts. A grid layout arranges its children in a grid, automatically calculating the size and placement of each child within the specified parameters.  The <code>cell-space-requirement</code> : init-keyword lets you specify the preferred space requirement for any individual cell in the grid layout.
Operations	None.
See also	<a href="#"><code>&lt;column-layout&gt;</code>, page 401</a> <a href="#"><code>&lt;row-layout&gt;</code>, page 423</a> <a href="#"><code>&lt;stack-layout&gt;</code>, page 433</a> <a href="#"><code>&lt;table-layout&gt;</code>, page 436</a>

	<i>Statement macro</i>
Summary	Lays out a series of gadgets horizontally.
Macro call	<code>horizontally ([options]) {panes}+ end</code>
Arguments	<code>options</code> Dylan arguments <sub>bnf</sub> . <code>panes</code> One or more occurrences of Dylan body <sub>bnf</sub> .
Values	None.
Description	<p>This macro lays a series of gadgets out horizontally, creating the necessary layouts for you automatically.</p> <p>The <code>options</code> are passed directly to the row layout, and thus can be any legitimate combinations of init-keywords for <code>&lt;row-layout&gt;</code>. If no options are specified, then the default values for row layout are used.</p>

The *panes* argument consists of a number of Dylan expressions, each of which creates an instance of a gadget or layout that is to be included in the horizontal layout.

**Example**

```
contain(horizontally ()
           make(<button>, label: "Hello");
           make(<button>, label: "World")
       end);
```

**See also**

<[row-layout
\[tabling\]\(#\), page 440  
\[vertically\]\(#\), page 442](#)

**<layout>**

*Open abstract class*

**Summary**

The superclass class of all layout classes.

**Superclasses**

<[sheet](#)>

**Init-keywords**

**space-requirement:**

An instance of type <[space-requirement](#)>. Required.

**width:** An instance of type <[integer](#)>. Required.

**height:** An instance of type <[integer](#)>. Required.

**min-width:** An instance of type <[integer](#)>. Default value: 0.

**min-height:** An instance of type <[integer](#)>. Default value: 0.

**max-width:** An instance of type <[integer](#)>. Default value: \$[fill](#).

**max-height:** An instance of type <[integer](#)>. Default value: \$[fill](#).

	<p><b>resizable?:</b> An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</p> <p><b>fixed-width?:</b> An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#f</code>.</p> <p><b>fixed-height?:</b> An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#f</code>.</p>
Description	<p>The class of layouts. This is the basic class from which all other forms of layout inherit. You cannot create direct instances of this class.</p> <p>The <b>space-requirement:</b> init-keyword describes the space required for the layout. It is generally calculated automatically based on the values of the various width and height init-keywords, and the class of layout that is being created.</p> <p>The <b>width:, height:, min-width:, min-height:, max-width:,</b> and <b>max-height:</b> init-keywords between them describe the configuration of the layout. The default values for these init-keywords (where applicable) are set such that the layout always fills the available space in any given direction.</p> <p>Finally, three init-keywords are available that control how the layout is affected when the frame containing it is resized. All three init-keywords take boolean values. You can specify whether a layout is resizable using the <b>resizable?:</b> init-keyword. If <b>fixed-width?:</b> or <b>fixed-height?:</b> are true, then the layout cannot be resized in the appropriate direction. Setting both to <code>#t</code> is equivalent to setting <b>resizable?:</b> to <code>#f</code>. Different subclasses of layout restrict the values of these init-keywords in different ways, such that, for instance, a row layout has a fixed height.</p>
Operations	None.
See also	<p><code>&lt;column-layout&gt;</code>, page 401</p> <p><code>&lt;grid-layout&gt;</code>, page 411</p>

`<pinboard-layout>`, page 421

`<row-layout>`, page 423

`<stack-layout>`, page 433

`<table-layout>`, page 436

## layout-border

*Generic function*

Summary	Returns the amount of whitespace around the children in a layout.	
Signature	<code>layout-border layout =&gt; border</code>	
Arguments	<code>layout</code>	An instance of type <code>type-union(&lt;row-layout&gt;, &lt;column-layout&gt;, &lt;table-layout&gt;, &lt;grid-layout&gt;, &lt;stack-layout&gt;).</code>
Values	<code>border</code>	An instance of type <code>&lt;integer&gt;</code> .
Description	Returns the amount of whitespace, in pixels, around the children in <i>layout</i> .  Note that this function does not apply to pinboard layouts, because the positioning of the children in a pinboard layout is completely in the control of the programmer.	
See also	<code>layout-border-setter</code> , page 415	

## layout-border-setter

*Generic function*

Summary	Sets the amount of whitespace around the children in a layout.	
Signature	<code>layout-border border layout =&gt; border</code>	

Arguments	<i>border</i>	An instance of type <integer>.
	<i>layout</i>	An instance of type type-union(<row-layout>, <column-layout>, <table-layout>, <grid-layout>, <stack-layout>).
Values	<i>border</i>	An instance of type <integer>.
Description		<p>Sets the amount of whitespace, in pixels, around the children in <i>layout</i>.</p> <p>You can also set this value a layout is created using the <b>border:</b> init-keyword.</p> <p>Note that this function does not apply to pinboard layouts, because the positioning of the children in a pinboard layout is completely in the control of the programmer.</p>
See also		<a href="#">layout-border</a> , page 415

<b>layout-equalize-heights?</b>		<i>Generic function</i>
Summary		Returns true if the children of the specified layout are all the same height.
Signature		<code>layout-equalize-heights? layout =&gt; equal?</code>
Arguments	<i>layout</i>	An instance of type type-union(<row-layout>, <column-layout>).
Values	<i>equal?</i>	An instance of type <boolean>.
Description		Returns true if the children of <i>layout</i> are all the same height. The layout must be either a row or a column layout.

You can only set this value when a layout is created, using the `equalize-widths?:` init-keyword. There is no equivalent setter function.

See also [layout-equalize-widths?, page 417](#)

## **layout-equalize-widths?** *Generic function*

Summary	Returns true if the children of the specified layout are all the same width.
Signature	<code>layout-equalize-widths? layout =&gt; equal?</code>
Arguments	<code>layout</code> An instance of type <code>type-union(&lt;row-layout&gt;, &lt;column-layout&gt;)</code> .
Values	<code>equal?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	<p>Returns true if the children of <code>layout</code> are all the same width. The layout must be either a row or a column layout.</p> <p>You can only set this value when a layout is created, using the <code>equalize-widths?:</code> init-keyword. There is no equivalent setter function.</p>

See also [layout-equalize-heights?, page 416](#)

## **<leaf-pane>** *Open abstract class*

Summary	The class of leaf panes.
Superclasses	<code>&lt;sheet&gt;</code>
Init-keywords	None.

Description	<p>The class of leaf panes. These are sheets that live at the leaf of the sheet tree that obeys the layout protocols.</p> <p>Subclass this class if you want to create a basic leaf pane.</p> <ul style="list-style-type: none"> <li>• If you want to do output to it, mix in one of the <code>&lt;sheet-with-medium-mixin&gt;</code> classes.</li> <li>• If you want to do input from it, mix in one of the <code>&lt;sheet-with-event-queue&gt;</code> classes.</li> <li>• If you want to repaint it, mix in one of the <code>&lt;sheet-with-repainting-mixin&gt;</code> classes.</li> </ul>
Operations	None.

<b>make</b>		<i>G.f. method</i>										
Summary	Creates an instance of <code>&lt;space-requirement&gt;</code> .											
Signature	<code>make space-requirement-class #key width min-width max-width height min-height max-height =&gt; space-req</code>											
Arguments	<p><i>space-requirement-class</i> The class <code>&lt;space-requirement&gt;</code>.</p> <table> <tr> <td><i>width</i></td><td>An instance of type <code>&lt;integer&gt;</code>. Default value: <code>\$fill</code>.</td></tr> <tr> <td><i>min-width</i></td><td>An instance of type <code>&lt;integer&gt;</code>. Default value: <i>width</i>.</td></tr> <tr> <td><i>max-width</i></td><td>An instance of type <code>&lt;integer&gt;</code>. Default value: <i>width</i>.</td></tr> <tr> <td><i>height</i></td><td>An instance of type <code>&lt;integer&gt;</code>. Default value: <code>\$fill</code>.</td></tr> <tr> <td><i>min-height</i></td><td>An instance of type <code>&lt;integer&gt;</code>. Default value: <i>height</i>.</td></tr> </table>		<i>width</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <code>\$fill</code> .	<i>min-width</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <i>width</i> .	<i>max-width</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <i>width</i> .	<i>height</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <code>\$fill</code> .	<i>min-height</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <i>height</i> .
<i>width</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <code>\$fill</code> .											
<i>min-width</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <i>width</i> .											
<i>max-width</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <i>width</i> .											
<i>height</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <code>\$fill</code> .											
<i>min-height</i>	An instance of type <code>&lt;integer&gt;</code> . Default value: <i>height</i> .											

	<i>max-height</i>	An instance of type <integer>. Default value: <i>height</i> .
Values	<i>space-req</i>	An instance of type <space-requirement>.
Description		Creates an instance of <space-requirement>. The various width and height arguments let you control the values of corresponding init-keywords to <space-requirement>, thereby control the width and height of a layout under various circumstances. See <space-requirement>, page 427, for a full description of this behavior.
See also		\$fill, page 410 <space-requirement>, page 427

## <multiple-child-composite-pane> *Open abstract class*

Summary	The class of composite panes that can have multiple children.
Superclasses	<layout>
Init-keywords	None.
Description	The class of composite panes that can have multiple children. Subclass this class if you want to create a class of pane that can have more than one child.
Operations	None.
See also	<single-child-composite-pane>, page 427

<null-pane>		<i>Sealed instantiable class</i>
Summary	The class of null panes.	
Superclasses	<leaf-pane>	
Init-keywords	None.	
Description	The class of null panes. This class acts as a filler: use it when you need to “fill space” somewhere in a complex layout.	
Operations	None.	
See also	<a href="#">&lt;spacing&gt;, page 569</a> <a href="#">with-spacing, page 613</a>	

<b>pane-display-function</b>		<i>Generic function</i>
Summary	Returns the function used to display the specified pane.	
Signature	<code>pane-display-function pane =&gt; pane-display-function</code>	
Arguments	<code>pane</code>	An instance of type <sheet>.
Values	<code>pane-display-function</code>	An instance of type <code>false-or(&lt;function&gt;)</code> .
Description	>Returns the function used to display <i>pane</i> , where <i>pane</i> is any pane that can have a <code>display-function:</code> init-keyword specified. The <i>value</i> returned by <code>pane-display-function</code> is the <i>value</i> of the <code>display-function:</code> init-keyword.  The display function gets called by the <code>handle-repaint</code> method for <simple-pane> and <drawing-pane>.	
See also	<a href="#">&lt;drawing-pane&gt;, page 409</a>	

**pane-layout***Generic function*

Summary	Returns the layout that contains the specified pane in <code>define pane</code> .	
Signature	<code>pane-layout pane =&gt; layout-pane</code>	
Arguments	<code>pane</code>	An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>layout-pane</code>	An instance of type <code>&lt;sheet&gt;</code> .
Description	Returns the layout that contains the specified pane in <code>define pane</code> .	
See also	<code>define pane</code> , page 406	

**<pinboard-layout>***Open abstract instantiable class*

Summary	The class of pinboard layouts.
Superclasses	<code>&lt;layout&gt;</code>
Init-keywords	<code>stretchable?:</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	The class of pinboard layouts. Unlike other types of layout, pinboard layouts are unusual in that the positioning and geometry of the children of a pinboard layout are entirely determined by the programmer. You can place children at any point in a pinboard layout, and the pinboard layout does not attempt to calculate an optimum position or size for any of them.  A pinboard layout leaves the subsequent positions of any children placed in the layout alone. However, the size of each child is constrained according to any constraints that have



**Figure 7.3** Three buttons arranged in a pinboard layout

been specified for those children. Compare this to fixed layouts, where the sizes of any children are not constrained in this way.

Because the size of a pinboard layout's children are constrained, pinboard layouts are most useful for placing sheets randomly in a layout, since DUIM ensures that the sheets remain a sensible size for their contents.

If `stretchable?:` is true, then the pinboard layout can be resized dynamically as its parent is resized (for instance, by the user resizing a window on screen).

Operations      None.

See also      `<fixed-layout>`, page 410

`<layout>`, page 413

## relayout-children

*Generic function*

Summary      Lays out the children of the specified sheet again.

Signature      `relayout-children sheet #key port-did-it? => ()`

Arguments      `sheet`      An instance of type `<sheet>`.

`port-did-it?`      An instance of type `<boolean>`. Default value: `#f`.

Values	None.
Description	Lays out the children of <i>sheet</i> again.
See also	<code>relayout-parent</code> , page 423

## **relayout-parent** *Generic function*

Summary	Lays out the parent of the specified sheet again.						
Signature	<code>relayout-parent sheet #key width height =&gt; ()</code>						
Arguments	<table> <tr> <td><i>sheet</i></td> <td>An instance of type <code>&lt;sheet&gt;</code>.</td> </tr> <tr> <td><i>width</i></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> <tr> <td><i>height</i></td> <td>An instance of type <code>&lt;integer&gt;</code>.</td> </tr> </table>	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .	<i>width</i>	An instance of type <code>&lt;integer&gt;</code> .	<i>height</i>	An instance of type <code>&lt;integer&gt;</code> .
<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .						
<i>width</i>	An instance of type <code>&lt;integer&gt;</code> .						
<i>height</i>	An instance of type <code>&lt;integer&gt;</code> .						
Values	None.						
Description	Lays out the parent of <i>sheet</i> again. If <i>width</i> and <i>height</i> are specified, then the parent is laid out in accordance with these dimensions.						
See also	<code>relayout-children</code> , page 422						

## **<row-layout>** *Open abstract instantiable class*

Summary	The class of row layouts.		
Superclasses	<code>&lt;layout&gt;</code>		
Init-keywords	<table> <tr> <td><code>border:</code></td> <td>An instance of type <code>&lt;integer&gt;</code>. Default value: 0.</td> </tr> </table>	<code>border:</code>	An instance of type <code>&lt;integer&gt;</code> . Default value: 0.
<code>border:</code>	An instance of type <code>&lt;integer&gt;</code> . Default value: 0.		

```

x-spacing:, spacing:
    An instance of type <integer>. Default
    value: 0.

equalize-heights?:
    An instance of type <boolean>. Default
    value: #f.

equalize-widths?:
    An instance of type <boolean>. Default
    value: #f.

y-alignment: An instance of type one-of(#"top",
    #"bottom", #"center"). Default value:
    #"top".

x-ratios:, ratios:
    An instance of type false-or(<sequence>).
    Default value: #f.

```

Description	The class of row layouts. A row layout arranges its children in a row, automatically calculating the size and placement of each child within the specified parameters.
-------------	--



**Figure 7.4** Three buttons arranged in a row layout

The **border:** init-keyword provides a border of whitespace around the children in the layout, and the value of this init-keyword represents the size of the border in pixels. This basically has the same effect as using the macro **with-spacing** around the layout, except it uses a simpler syntax.

The **spacing:** or **x-spacing:** init-keywords let you specify how much horizontal space, in pixels, should be inserted between the children of the layout. These two init-keywords can be used interchangeably.

If true, **equalize-heights?:** ensures that all the children of the layout have the same height.

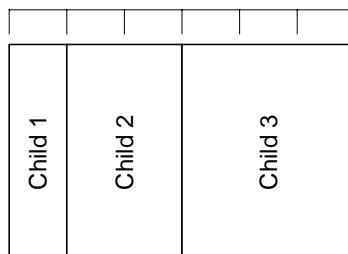
If true, **equalize-widths?:** ensures that all the children of the layout have the same width.

By default, all the children of a row layout are aligned at the top. You can specify that they should be aligned at the bottom, or in the center, using the **y-alignment:** keyword.

The **ratios:** or **x-ratios:** init-keywords let you specify the proportion of the total layout that should be taken up by each individual child. These two init-keywords can be used interchangeably.

The value passed to **ratios:** needs to be a sequence of as many integers as there are children in the layout. Each child is then allocated the appropriate portion of horizontal space in the layout. For example, if the value #(1, 2, 3) is specified for the **ratios:** init-keyword of a row layout containing three children, then the first child would claim a sixth of the available horizontal space, the second child would claim a third of the horizontal space, and the third child would claim half the horizontal space, as shown in the diagram below.

**ratios: #(1, 2, 3)**



Operations	None.
Example	To make a row of buttons that are all the same size:  <pre>contain(make(&lt;row-layout&gt;,             equalize-widths?: #t,             children: buttons))</pre>
See also	<a href="#">&lt;column-layout&gt;</a> , page 401 <a href="#">horizontally</a> , page 412 <a href="#">&lt;layout&gt;</a> , page 413 <a href="#">&lt;grid-layout&gt;</a> , page 411 <a href="#">&lt;stack-layout&gt;</a> , page 433 <a href="#">&lt;table-layout&gt;</a> , page 436

## <simple-pane> *Open abstract instantiable class*

Summary	The class of simple panes.
Superclasses	<a href="#">&lt;layout&gt;</a>
Init-keywords	<b>display-function:</b> An instance of type <code>false-or(&lt;function&gt;)</code> . Default value: <code>#f</code> .
Description	The class of simple panes. The <b>display-function:</b> init-keyword defines the display function for the pane. This gets called by the <b>handle-repaint</b> method for <a href="#">&lt;simple-pane&gt;</a> .
Operations	None.
See also	<a href="#">&lt;drawing-pane&gt;</a> , page 409 <a href="#">handle-repaint</a> , page 253

[pane-display-function](#), page 420

## <single-child-composite-pane>

*Open abstract class*

Summary      The class of composite panes that can only have one child.

Superclasses    <layout>

Init-keywords   None.

Description      The class of composite panes that can only have one child.

Operations     None.

See also       <multiple-child-composite-pane>, page 419

## <space-requirement>

*Abstract instantiable class*

Summary      The class of all space requirement objects.

Superclasses   <object>

Init-keywords   **width:**      An instance of type <integer>. Default value: \$fill.

**min-width:**    An instance of type <integer>. Default value: *width*.

**max-width:**    An instance of type <integer>. Default value: *width*.

**height:**       An instance of type <integer>. Default value: \$fill.

**min-height:**   An instance of type <integer>. Default value: *height*.

	<b>max-height:</b> An instance of type <code>&lt;integer&gt;</code> . Default value: <code>height</code> .
	<b>label:</b> An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code> .
Description	<p>The class of all space requirement objects. This type of object is used to reserve space when it is required in a layout in order to accommodate gadgets or other layouts.</p> <p>The various init-keywords let you constrain the width and height of the object in a variety of ways.</p> <p>If no init-keywords are specified, the object returned tries to fill all the available space.</p> <p>Specifying <code>width:</code> or <code>height:</code> specifies the preferred width or height of the object.</p> <p>Specifying any of the <code>min-</code> or <code>max-</code> init-keywords lets you minimum and maximum width or height for the object.</p> <p>The following inequalities hold for all widths and heights:</p> <pre><code>min-height: &lt;= height: &lt;= max-height: min-width: &lt;= width: &lt;= max-width:</code></pre> <p>If either <code>min-width:</code> or <code>min-height:</code> is 0, the object is “infinitely shrinkable” in that direction. If either <code>max-width:</code> or <code>max-height:</code> is <code>\$fill</code>, the object is “infinitely stretchable” in that direction. The latter is a particularly useful way of ensuring that objects fill the available width, and can be used, say, to ensure that a series of buttons fill the entire width of the layout that they occupy.</p> <p>An example of the use of <code>max-width:</code> to force the size of a button to fit the available space can be found in the entry for <code>&lt;button&gt;</code>, page 470.</p> <p>The <code>label:</code> init-keyword specifies a label which is measured to give the preferred width and height.</p>

**Operations** `space-requirement-height space-requirement-max-height space-requirement-max-width space-requirement-min-height space-requirement-min-width space-requirement-width`

**Example** Given the following definition of a button class:

```
define class <basic-test-button>
  (<leaf-pane>)
end class <basic-test-button>;
```

The following method for `do-compose-space` creates the necessary space requirements to accommodate the new button class in a layout.

```
define method do-compose-space
  (pane :: <basic-test-button>, #key width, height)
=> (space-req :: <space-requirement>)
  ignore(width, height);
  make(<space-requirement>,
    width: 40,
    height: 15)
end method do-compose-space;
```

**See also** `$fill`, page 410

## space-requirement?

## Generic function

**Summary** Returns true if the specified object is a space requirement.

**Signature** `space-requirement? object => boolean`

**Arguments** `object` An instance of type `<object>`.

**Values** `boolean` An instance of type `<boolean>`.

**Description** Returns true if `object` is an instance of `<space-requirement>`.

**See also** `<space-requirement>`, page 427

## space-requirement-height *Generic function*

Summary	Returns the preferred height of the specified space requirement.	
Signature	<code>space-requirement-height sheet space-req =&gt; height</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>space-req</code>	An instance of type <code>&lt;space-requirement&gt;</code> .
Values	<code>height</code>	An instance of type <code>&lt;number&gt;</code> .
Description	Returns preferred the height of <code>space-req</code> . This is the value of the <code>height:</code> init-keyword that was passed when the object was created.	
See also	<a href="#">space-requirement-max-height, page 430</a> <a href="#">space-requirement-min-height, page 431</a>	

## space-requirement-max-height *Generic function*

Summary	Returns the maximum allowed height of the specified space requirement.	
Signature	<code>space-requirement-max-height sheet space-req =&gt; max-height</code>	
Arguments	<code>sheet</code>	An instance of type <code>&lt;sheet&gt;</code> .
	<code>space-req</code>	An instance of type <code>&lt;space-requirement&gt;</code> .
Values	<code>max-height</code>	An instance of type <code>&lt;number&gt;</code> .
Description	Returns the maximum allowed height of <code>space-req</code> . This is the value of the <code>max-height:</code> init-keyword that was passed when the object was created.	

See also      [space-requirement-height, page 430](#)  
[space-requirement-min-height, page 431](#)

## space-requirement-max-width                          *Generic function*

Summary      Returns the maximum allowed width of the specified space requirement.

Signature     `space-requirement-max-width sheet space-req => max-width`

Arguments    `sheet`               An instance of type `<sheet>`.  
               `space-req`           An instance of type `<space-requirement>`.

Values        `max-width`       An instance of type `<number>`.

Description    Returns the maximum allowed width of `space-req`. This is the value of the `max-width: init`-keyword that was passed when the object was created.

See also      [space-requirement-min-width, page 432](#)  
[space-requirement-width, page 432](#)

## space-requirement-min-height                          *Generic function*

Summary      Returns the minimum allowed height of the specified space requirement.

Signature     `space-requirement-min-height sheet space-req => min-height`

Arguments    `sheet`               An instance of type `<sheet>`.  
               `space-req`           An instance of type `<space-requirement>`.

Values        `min-height`       An instance of type `<number>`.

Description	Returns the minimum allowed height of <i>space-req</i> . This is the value of the <code>min-height:</code> init-keyword that was passed when the object was created.
See also	<code>space-requirement-height</code> , page 430 <code>space-requirement-max-height</code> , page 430

## space-requirement-min-width *Generic function*

Summary	Returns the minimum allowed width of the specified space requirement.				
Signature	<code>space-requirement-min-width sheet space-req =&gt; min-width</code>				
Arguments	<table> <tr> <td><i>sheet</i></td> <td>An instance of type <code>&lt;sheet&gt;</code>.</td> </tr> <tr> <td><i>space-req</i></td> <td>An instance of type <code>&lt;space-requirement&gt;</code>.</td> </tr> </table>	<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .	<i>space-req</i>	An instance of type <code>&lt;space-requirement&gt;</code> .
<i>sheet</i>	An instance of type <code>&lt;sheet&gt;</code> .				
<i>space-req</i>	An instance of type <code>&lt;space-requirement&gt;</code> .				
Values	<i>min-width</i> An instance of type <code>&lt;number&gt;</code> .				
Description	Returns the minimum allowed width of <i>space-req</i> . This is the value of the <code>min-width:</code> init-keyword that was passed when the object was created.				
See also	<code>space-requirement-max-width</code> , page 431 <code>space-requirement-width</code> , page 432				

## space-requirement-width *Generic function*

Summary	Returns the preferred width of the specified space requirement.
Signature	<code>space-requirement-width sheet space-req =&gt; width</code>
Arguments	<i>sheet</i> An instance of type <code>&lt;sheet&gt;</code> .

	<i>space-req</i>	An instance of type <code>&lt;space-requirement&gt;</code> .
Values	<i>width</i>	An instance of type <code>&lt;number&gt;</code> .
Description		Returns the preferred width of <i>space-req</i> . This is the value of the <code>width:</code> init-keyword that was passed when the object was created.
See also		<code>space-requirement-max-width</code> , page 431 <code>space-requirement-min-width</code> , page 432

## **<stack-layout>** *Open abstract instantiable class*

Summary	The class of stack layouts.
Superclasses	<code>&lt;layout&gt;</code>
Init-keywords	<b>border:</b> An instance of type <code>&lt;integer&gt;</code> . Default value: 0. <b>mapped-page:</b> An instance of <code>&lt;sheet&gt;</code> .
Description	The class of stack layouts. Stack layouts position all of their children at the top-left one on top of the other. The layout sizes itself to be large enough to fit the largest child. They are primarily useful for creating layouts that simulate sets of several pages where only one child is visible at a time, and all the others are withdrawn, and are used to control the layout of elements such as tab controls or wizard frames. To make a new page appear, you withdraw the current page, and then map the new page. The new page is automatically the correct size and in the correct position.

The **border:** init-keyword provides a border of whitespace around the children in the layout, and the value of this init-keyword represents the size of the border in pixels. This basically has the same effect as using the macro **with-spacing** around the layout, except it uses a simpler syntax.

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.

Operations	None.
See also	<a href="#"><code>&lt;column-layout&gt;</code></a> , page 401 <a href="#"><code>&lt;grid-layout&gt;</code></a> , page 411 <a href="#"><code>&lt;layout&gt;</code></a> , page 413 <a href="#"><code>&lt;row-layout&gt;</code></a> , page 423 <a href="#"><code>&lt;table-layout&gt;</code></a> , page 436

	<i>Generic function</i>
Summary	Returns the currently mapped page for a stack layout.
Signature	<code>stack-layout-mapped-page <i>stack-layout</i> =&gt; <i>page</i></code>
Arguments	<code>stack-layout</code> An instance of <code>&lt;stack-layout&gt;</code> .
Values	<code>page</code> An instance of <code>&lt;sheet&gt;</code> .
Description	Returns the currently mapped <i>page</i> for the specified <i>stack-layout</i> .

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

Summary	Sets the mapped page for a stack layout.	
Signature	<code>stack-layout-mapped-page page stack-layout =&gt; page</code>	
Arguments	<i>page</i>	An instance of <code>&lt;sheet&gt;</code> .
	<i>stack-layout</i>	An instance of <code>&lt;stack-layout&gt;</code> .
Values	<i>page</i> An instance of <code>&lt;sheet&gt;</code> .	
Description	Sets the mapped page for the specified <i>stack-layout</i> to <i>page</i> .	

**table-contents** *Generic function*

Summary	Returns the contents of the specified table.	
Signature	<code>table-contents table =&gt; contents</code>	
Arguments	<i>table</i>	An instance of type <code>&lt;table-layout&gt;</code> .
Values	<i>contents</i>	An instance of type <code>&lt;sheet&gt;</code> .
Description	Returns the contents of <i>table</i> .	
See also	<a href="#">table-contents-setter</a> , page 435	

**table-contents-setter** *Generic function*

Summary	Sets the contents of the specified table.	
Signature	<code>table-contents-setter contents table =&gt; contents</code>	
Arguments	<i>contents</i>	An instance of type <code>&lt;sheet&gt;</code> .

	<i>table</i>	An instance of type < <b>table-layout</b> >.
Values	<i>contents</i>	An instance of type < <b>sheet</b> >.
Description		Sets the contents of <i>table</i> .
See also		<b>table-contents</b> , page 435

## **<table-layout>** *Open abstract instantiable class*

Summary	The class of table layouts.
Superclasses	< <b>layout</b> >
Init-keywords	<b>border:</b> An instance of type < <b>integer</b> >. Default value: 0. <b>rows:</b> An instance of type <b>false-or(&lt;integer&gt;)</b> . Default value: #f. <b>columns:</b> An instance of type <b>false-or(&lt;integer&gt;)</b> . Default value: #f. <b>contents:</b> An instance of type <b>limited(&lt;sequence&gt;, of: limited(&lt;sequence&gt;, of: &lt;sheet&gt;))</b> . <b>x-spacing:</b> An instance of type < <b>integer</b> >. Default value: 0. <b>y-spacing:</b> An instance of type < <b>integer</b> >. Default value: 0. <b>x-ratios:</b> An instance of type <b>false-or(&lt;sequence&gt;)</b> . Default value: #f. <b>y-ratios:</b> An instance of type <b>false-or(&lt;sequence&gt;)</b> . Default value: #f.

**x-alignment:** An instance of type `one-of(#"left",  
#"right", #"center")`. Default value:  
#"left".

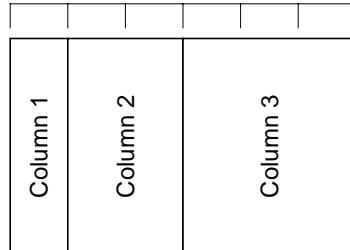
**y-alignment:** An instance of type `one-of(#"top",  
#"bottom", #"center")`. Default value:  
#"top".

Description	<p>The class of table layouts.</p> <p>The <b>border:</b> init-keyword provides a border of whitespace around the children in the layout, and the value of this init-keyword represents the size of the border in pixels. This basically has the same effect as using the macro <code>with-spacing</code> around the layout, except it uses a simpler syntax.</p> <p>The <b>rows:</b> and <b>columns:</b> init-keywords are used to specify the number of rows and columns for the table layout.</p> <p>The <b>contents:</b> init-keyword is used to specify the contents of each cell of the table. It should consist of a sequence of sequences of sheets. If <b>contents:</b> is not specified, you should supply the children of the table with a number of rows and columns. You should not supply both children and rows and columns, however.</p> <p>The <b>x-spacing:</b> and <b>y-spacing:</b> init-keywords let you specify how much vertical and horizontal space should be inserted, in pixels, between the children of the layout.</p> <p>The <b>x-ratios:</b> and <b>y-ratios:</b> init-keywords let you specify the proportion of the total horizontal and vertical space that should be taken up by each individual child.</p> <p>The value passed to <b>x-ratios:</b> needs to be a sequence of as many integers as there are columns of children in the layout. The value passed to <b>y-ratios:</b> needs to be a sequence of as many integers as there are rows of children in the layout. Each child is then allocated the appropriate portion of horizontal and vertical space in the layout, according to the combination of the values for these two keywords.</p>
-------------	---

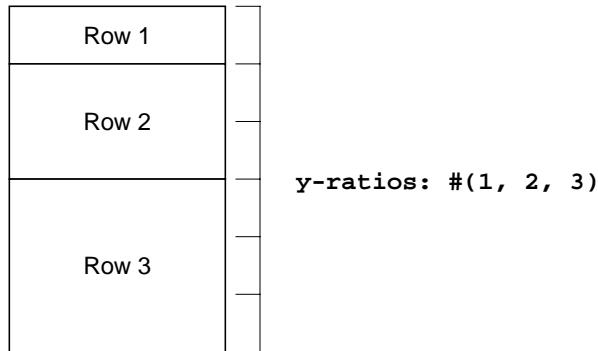
The two init-keywords can be used on their own, or together, as described in the examples below.

For example, if the value `#{1, 2, 3}` is specified for the `x-ratios:` init-keyword of a table layout containing three columns of children, then the first column would claim a sixth of the available horizontal space, the second column would claim a third of the horizontal space, and the third column would claim half the horizontal space, as shown in the diagram below.

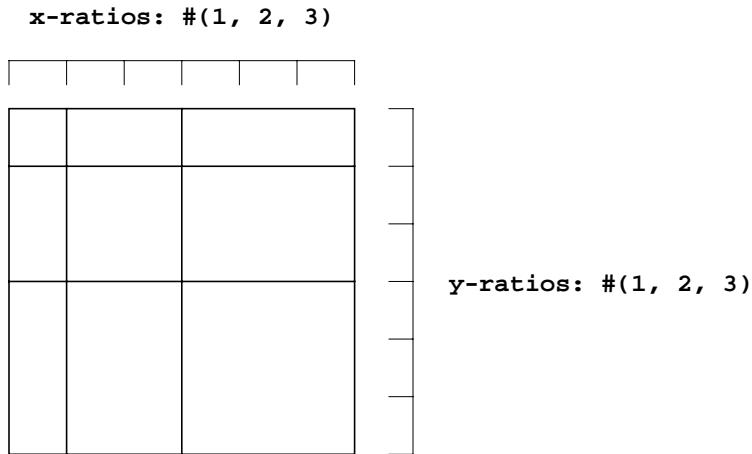
`x-ratios: #{1, 2, 3}`



Alternatively, if the value `#{1, 2, 3}` is specified for the `y-ratios:` init-keyword of a table layout containing three rows of children, then the first row would claim a sixth of the available vertical space, the second row would claim a third of the vertical space, and the third row would claim half the vertical space, as shown in the diagram below.



Finally, if both the **x-ratios:** and **y-ratios:** init-keywords are specified, then each child in the layout is affected individually, as shown in the diagram below.



By default, all the children of a table layout are left-aligned. You can specify that they should be right or center-aligned using the **x-alignment:** keyword.

By default, all the children of a table layout are aligned at the top. You can specify that they should be aligned at the bottom, or in the center, using the **y-alignment:** keyword.

Operations	<code>table-contents table-contents-setter</code>
Example	<pre>*t* := make(&lt;vector&gt;, size: 9); for (i from 1 to 9) *t*[i - 1] :=     make(&lt;button&gt;, label: format-to-string("%d", i)) end;  contain(make(&lt;table-layout&gt;,             x-spacing: 10, y-spacing: 0,             children: *t*, columns: 3));</pre>
See also	<p><code>&lt;column-layout&gt;</code>, page 401</p> <p><code>&lt;grid-layout&gt;</code>, page 411</p> <p><code>&lt;layout&gt;</code>, page 413</p> <p><code>&lt;row-layout&gt;</code>, page 423</p> <p><code>&lt;stack-layout&gt;</code>, page 433</p> <p><code>tabling</code>, page 440</p>

<b>tabling</b> <i>Statement macro</i>					
Summary	Lays out a series of gadgets in a table.				
Macro call	<code>tabling ([<i>options</i>]) {<i>panes</i>}+ end</code>				
Arguments	<table> <tr> <td><i>options</i></td><td>Dylan arguments<sub>bnf</sub>.</td></tr> <tr> <td><i>panes</i></td><td>One or more occurrences of Dylan body<sub>bnf</sub>.</td></tr> </table>	<i>options</i>	Dylan arguments <sub>bnf</sub> .	<i>panes</i>	One or more occurrences of Dylan body <sub>bnf</sub> .
<i>options</i>	Dylan arguments <sub>bnf</sub> .				
<i>panes</i>	One or more occurrences of Dylan body <sub>bnf</sub> .				
Values	None.				
Description	<p>This macro lays a series of gadgets out in a table, creating the necessary layouts for you automatically.</p> <p>The <i>options</i> are passed directly to the table layout, and thus can be any legitimate combinations of init-keywords for <code>&lt;table-layout&gt;</code>. If no options are specified, then the default values for table layout are used.</p>				

The *panes* argument consists of a number of Dylan expressions, each of which creates an instance of a gadget or layout that is to be included in the vertical layout.

See also [horizontally](#), page 412  
[<table-layout>](#), page 436  
[vertically](#), page 442

## <top-level-sheet> *Open abstract instantiable class*

Summary	The class of top level sheets.
Superclasses	<layout>
Init-keywords	<b>display:</b> An instance of type <code>false-or(&lt;display&gt;)</code> . Default value: <code>#f</code> .
	<b>frame:</b> An instance of type <code>false-or(&lt;frame&gt;)</code> . Default value: <code>#f</code> .
	<b>frame-manager:</b> An instance of type <code>false-or(&lt;frame-manager&gt;)</code> . Default value: <code>#f</code> .
	<b>container:</b> An instance of type <code>false-or(&lt;object&gt;)</code> . Default value: <code>#f</code> .
	<b>container-region:</b> An instance of type <code>false-or(&lt;region&gt;)</code> . Default value: <code>#f</code> .
Description	The class of top level sheets.  The <code>container:</code> and <code>container-region:</code> init-keywords are for use in embedded frames, such as OLE objects in HTML browser windows. The <code>container:</code> init-keyword denotes the container itself, and <code>container-region:</code> is used to specify

the region of the screen in which the container appears. Note that the container referred to is a native window system object.

Operations      None.

	<i>Statement macro</i>				
<b>vertically</b>					
Summary	Lays out a series of gadgets vertically.				
Macro call	<code>vertically ([<i>options</i>]) {<i>panes</i>}+ end</code>				
Arguments	<table border="0"> <tr> <td><i>options</i></td><td>Dylan arguments<sub>bnf</sub>.</td></tr> <tr> <td><i>panes</i></td><td>One or more occurrences of Dylan body<sub>bnf</sub>.</td></tr> </table>	<i>options</i>	Dylan arguments <sub>bnf</sub> .	<i>panes</i>	One or more occurrences of Dylan body <sub>bnf</sub> .
<i>options</i>	Dylan arguments <sub>bnf</sub> .				
<i>panes</i>	One or more occurrences of Dylan body <sub>bnf</sub> .				
Values	None.				
Description	<p>This macro lays a series of gadgets out vertically, creating the necessary column layout for you automatically.</p> <p>The <i>options</i> are passed directly to the column layout, and thus can be any legitimate combinations of init-keywords for <code>&lt;column-layout&gt;</code>. If no options are specified, then the default values for table layout are used.</p> <p>The <i>panes</i> argument consists of a number of Dylan expressions, each of which creates an instance of a gadget or layout that is to be included in the vertical layout.</p>				
Example	<pre>contain(vertically (border: 5, equalize-widths: #t)         make(&lt;button&gt;, label: "Hello");         make(&lt;button&gt;, label: "World")       end);</pre>				
See also	<p><code>&lt;column-layout&gt;</code>, page 401</p> <p><code>horizontally</code>, page 412</p> <p><code>tabling</code>, page 440</p>				





# 8

---

---

# DUIM-Gadgets Library

## 8.1 Overview

The elements that comprise a Graphical User Interface (GUI) are arranged in a hierarchical ordering of object classes. At the top level of the DUIM hierarchy there are three main classes, `<sheet>`, `<gadget>`, and `<frame>`, all of which are subclasses of `<object>`.

The DUIM-Gadgets library contains classes that define a wide variety of gadgets for use in your GUI applications, such as push buttons, radio buttons, and check boxes. The library also provides the necessary functions, generic functions, and macros for creating and manipulating these classes. The library contains a single module, `duim-gadgets`, from which all the interfaces described in this chapter are exposed. Section 8.11 on page 465 contains complete reference entries for each exposed interface.

Gadgets are the basic behavioral GUI element (above the level of events).

- Gadgets do not *need* to have a visual presence, though in practice every gadget provided by DUIM does, since all general instances of `<gadget>` are also general instances of `<sheet>`.
- Many classes of gadget maintain some kind of state for their behavior, and in practice some of this is usually reflected in the UI. For example, you can tell that a check box is selected just by looking at it.

- They handle events and turn these into callbacks, for convenience.

Some of the more important types of gadget are as follows:

**Buttons** A wide variety of buttons are provided by DUIM. These include not only standard buttons such as push buttons and radio buttons, but items that can be placed within menus.

**Action gadgets** An action gadget is any gadget that can be used to perform an action, such as a button, or menu command.

**Value gadgets** A value gadget is any gadget that can have a value associated with it. In principle, this is true of the majority of gadgets, but the value of a gadget is more important for certain types of gadget (for instance, lists or radio boxes) than for others (for instance, push buttons).

### **Value range gadgets**

Value range gadgets are those value gadgets for which the possible value sits within a defined range. This includes gadgets such as scroll bars and sliders.

### **Collection gadgets**

Collection gadgets are those gadgets that can contain a number of “child” gadgets, the specification of which can be described in terms of a Dylan collection, and includes gadgets such as list controls and groups of buttons. Usually, the behavior of each of the “child” gadgets is interdependent in some way; for example, only one button in a group of radio buttons may be selected at any time. With collection gadgets, you can specify the “child” gadgets very simply, without having to worry about defining each “child” explicitly.

Each of these types of gadget is described in more detail in subsequent sections, and full reference entries for every interface exposed in the DUIM-Gadgets library are available in Section 8.11 on page 465. For a more general

introduction to the gadgets provided in DUIM, see the tour in the *Building Applications using DUIM* book. See the same book for a more practical example of implementing an application using the DUIM library.

## 8.2 Callbacks and keys

When an event occurs in a user interface (for example, a button is pressed, a menu command is chosen, or an item in a list is double-clicked), you usually want some operation to be performed. If the user of your application chooses the **File > Open** command, a File Open dialog should be displayed. If the user clicks on an **OK** button in a dialog, the dialog should be dismissed and the appropriate changes to the application state to be performed. In DUIM, you can provide this functionality by specifying a function known as a *callback*.

Generally speaking, a callback gets passed a single argument, which is the gadget that is affected. Thus, the argument passed to the callback for a button is the button itself. Callbacks do not need to have a return value, although they are not forbidden either. If a value is returned by a callback function, then it is just ignored.

Callbacks are used in preference to event handlers because Dylan does not let you write methods that specialize on individual instances. In languages such as C, you uniquely name each element in an interface, and then provide behavior for each element by writing event handlers that contain case statements that let you discriminate on individual elements. This is a somewhat inelegant solution. Instead, in Dylan you specify the names of the callbacks for each element in an interface when you *create* the elements. It is then a simple matter for the system to know what behavior goes with what elements, and is much less tedious than having to write many cumbersome methods for `handle-event`.

In Dylan, you use events in order to create new kinds of class. If you were creating a new kind of button, you would need to define a new method for `handle-event` in order to describe what happens when you click on an instance of that button. You would then write callbacks to deal with particular instance of the new class of button.

By contrast with callbacks, you can also provide functions in DUIM known as **keys**, which are specific to collection gadgets. A key is used to set the value of some aspect of the collection gadget for which the key is defined. With keys, therefore, the values returned by the function are fundamental to the operation of the gadget. There are two keys that are generally used by gadgets, known as the **value** key and the **label** key. The **value** key is a function that is used to calculate the value of the gadget for which the key is defined. The **label** key is used to calculate the printed representation, or label, of all the items in a collection gadget.

## 8.3 Gadget protocols

Gadgets are objects that make up an interface: the menus, buttons, sliders, check lists, tool bars, menu bars, and so on. Gadget classes may support three protocols, **value**, **items**, and **activate**.

- Gadgets that support the **value** protocol respond to the **gadget-value** message, a value-changed callback, and have a setter function associated with them.
- Gadgets that support the **items** protocol respond to **gadget-items** and have a gadget setter function associated with them.
- Gadgets that support the **activate** protocol have an activation callback associated with them.

Gadgets have a set of slots, or properties, associated with them: **gadget-label**, **gadget-value**, **gadget-items**, and **gadget-enabled?**. Every gadget has some or all of these properties.

- |                     |   |
|---------------------|---|
| <b>gadget-label</b> | This slot holds the label that appears on the gadget on the screen. If a gadget does not have a label, the <b>gadget-label</b> function returns <b>#f</b> . |
| <b>gadget-value</b> | This slot holds the value(s) of the gadget. If a gadget does not have any values, the <b>gadget-value</b> function returns <b>#f</b> .                      |
| <b>gadget-items</b> | This slot is a list of the contents of the gadget. If the gadget does not have items, for example a button, <b>gadget-items</b> returns nothing.            |

`gadget-enabled?`

This slot tests whether or not the gadget is active. All gadgets have a `gadget-enabled?` slot.

An introduction to the protocols supported by different sorts of gadget can also be found in the *Building Applications using DUIM* book.

## 8.4 The class hierarchy for DUIM-Gadgets

This section presents an overview of the available classes of gadget, and describes the class hierarchy present.

In each table below, classes that support the `items` protocol are displayed in **bold text**, and classes that support the activate protocol are displayed using *italic text*.

**Note:** In Table 8.5, every subclass shown supports the `items` protocol, though for clarity, no bold is used.

All subclasses of `<value-gadget>` support the `value` protocol. These are described in Section 8.4.2, Section 8.4.4, and Section 8.4.5.

### 8.4.1 The `<gadget>` class and its subclasses

The base class for the majority of DUIM gadgets is the `<gadget>` class, which is itself a subclass of `<object>`. All other DUIM gadgets are subclasses of `<gadget>`, with the exception of `<list-item>`, `<tree-node>`, and `<table-item>`.

The immediate subclasses of `<gadget>` are shown in Table 8.1. Only `<value-gadget>` and `<page>` have any subclasses defined. See Section 8.4.2 on page 451 and Section 8.4.2 on page 451 for details of these subclasses.

The `<gadget>` class provides a number of subclasses that allow particular parts of a user interface to be created:

`<menu>`

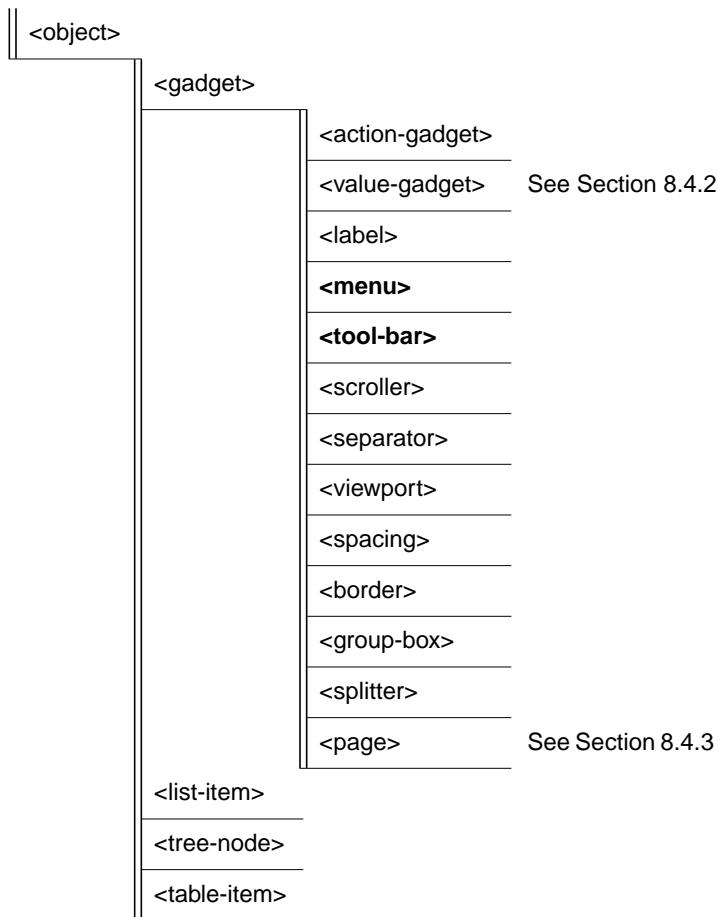
Use this class to add a menu to the menu bar of any application frame. Menus themselves contain commands created using the menu-specific button and collection gadgets described in Section 8.4.4 and Section 8.4.5.

- <**tool-bar**> This class is used to add a tool bar to an application frame. A tool bar is a row of buttons that duplicates the functionality of the most commonly used menu commands, thereby providing the user with quick access to the most useful operations in the application.
- <**scroller**> This is a generic scrolling gadget that can be used in a number of situations.
- <**viewport**> A viewport can be used to create a generic pane for displaying specialized contents that you may have defined. Use this class when there is no other class provided for displaying the objects in question.
- <**splitter**> This class can be used to split the current view in half. This allows the user, for example, to create a second view of the same document.

The <**gadget**> class provides a number of subclasses that allow general spatial and grouping capability, in addition to the layout functionality described in Chapter 7, “DUIM-Layouts Library”. These are as follows:

- <**label**> This class is used to assign label to many other types of gadget. Many gadgets can be assigned one or more labels, usually by means of a label: init-keyword. This class is used to assign any label.
- <**separator**> This allows a line to be drawn between any two gadgets or groups of gadgets, so as to provide a visible barrier between them.
- <**spacing**> This allows you to specify how much space should be placed between any two gadgets or groups of gadgets.
- <**border**> This allows a visible border to be placed around any number of gadgets.
- <**group-box**> This allows you to group together any number of related gadgets in a frame. Grouped elements are usually displayed with a border and label identifying the grouping.

**Table 8.1** Overall class hierarchy for the DUIM-Gadgets library



#### 8.4.2 Subclasses of <value-gadget>

Any gadget that can take a value of some sort is a subclass of `<value-gadget>`. As might be expected, this includes the majority of the gadgets in the DUIM-Gadgets library.

Every subclass of `<value-gadget>` supports the `value` protocol, as described in Section 8.1 on page 445.

Several subclasses of <**value-gadget**> themselves have a number of subclasses defined. These include:

<**text-gadget**> Any gadget into which you can type text. These include both text editors (multiple line edit controls) and text fields (single line edit controls).

<**value-range-gadget**>

Value gadgets whose value can vary within a known range, such as scroll bars.

<**button**>

Any button, such as a radio button, check button, or push button. See Section 8.4.4 on page 455 for more details about the classes of button available.

<**collection-gadget**>

Any gadget whose contents form a collection, such as a list, a tree control, or a group of buttons. See Section 8.4.5 on page 457 for more details about the classes of collection gadget available.

Also provided are the following specific GUI elements:

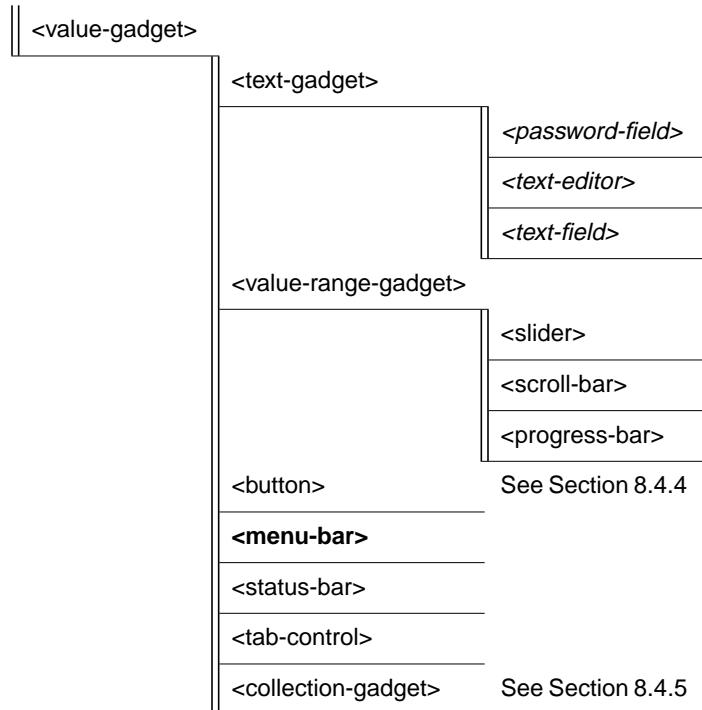
<**menu-bar**> This used to create the standard menu bar that is commonly found across the top of an application frame.

<**status-bar**> This is used to create a status bar, usually placed at the bottom of an application frame. A status bar is used to display miscellaneous information about the current state of the application.

<**tab-control**> Tab controls are analogous to dividers in a filing cabinet or notebook, with multiple logical pages of information displayed within the same window. Clicking on any of the tabs displayed in a tab control displays a new page of information.

The subclasses of <value-gadget> are as shown in Table 8.2.

**Table 8.2** Subclasses of the <value-gadget> class

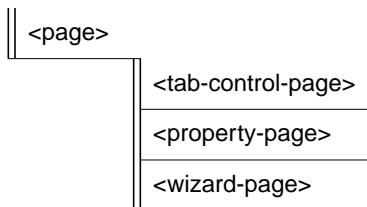


### 8.4.3 Subclasses of <page>

The `<page>` class is the base class of gadgets that are used to display a whole page of information within a “parent” element, with the page itself optionally containing other layouts or gadgets. Pages are used in situations where different sets of information (the pages themselves) need to be displayed in a common parent.

The subclasses of `<page>` are as shown in Table 8.3.

**Table 8.3** Subclasses of the `<page>` class



The `<tab-control-page>` class is used to define the elements that are associated with each tab in a tab control.



**Figure 8.1** A tab control page

The `<property-page>` class performs a similar job for property frames (visually, a property frame looks like a tab control in a dialog box, and is one way of implementing a dialog box that has several pages of information. Property frames are so named because they are often used to display the user-configurable properties of an application.



**Figure 8.2** A property page

The `<wizard-page>` class is used to define the elements in each page of a wizard frame. Wizard frames are another form of multi-page dialog, but consist of several physically distinct windows that are presented to the user in a strict order.



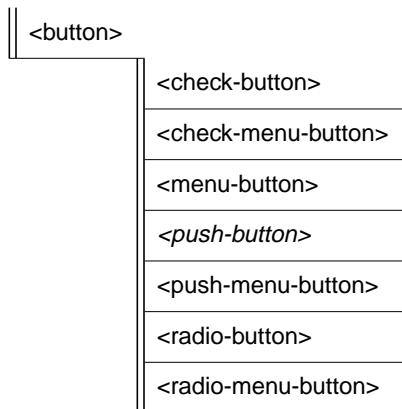
**Figure 8.3** A tab control page

#### 8.4.4 Subclasses of `<button>`

The subclasses of `<button>` are as shown in Table 8.4. These subclasses include not only buttons that can appear in any sheet, but also their equivalent classes of menu item. Thus, an instance of `<check-button>` represents a button whose state can toggle a specific value on and off, and an instance of `<check-menu-button>` represents a menu item whose state can toggle a specific value on and off in the same way.

Since all the subclasses of `<button>` are themselves value gadgets, each one supports the `value` protocol, as described in Section 8.1 on page 445.

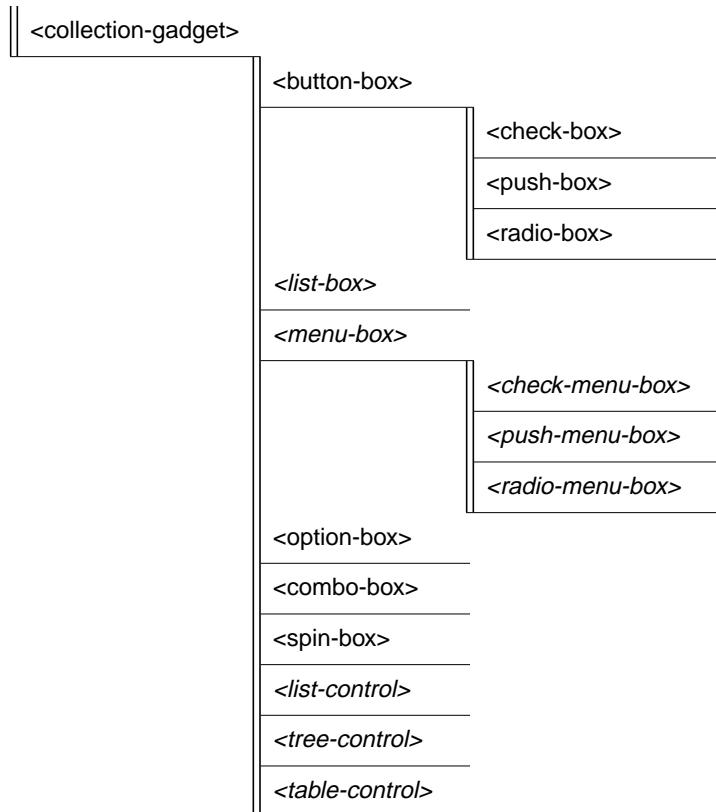
**Table 8.4** Subclasses of the `<button>` class



### 8.4.5 Subclasses of <collection-gadget>

The subclasses of <collection-gadget> are as shown in Table 8.5. All of these subclasses support the `items` protocol, even though they are not displayed in bold.

**Table 8.5** Subclasses of the <collection-gadget> class



Two subclasses themselves have a number of subclasses defined: those subclasses representing collections of buttons:

<**button-box**> These are used to create collections of buttons of the same type. You can create collections of any of the three basic types of button available: check buttons, radio buttons, or push buttons.

**<menu-box>** These are used to create collections of menu items of the same type. As with **<button-box>**, you can create collections of any of the three basic types of menu button available: check, radio, or push menu buttons.

In addition, the following types of list are provided:

**<list-box>** These are standard list boxes, allowing a list of items to be displayed in a pane, with a scroll bar allowing the complete list to be viewed if necessary. List boxes may be single, multiple, or no selection.

**<option-box>** A standard drop-down list box. This is similar to a list box, except that the entire list of options is only displayed on demand. In its closed state, only the current selection is visible.

**<combo-box>** A combo box combines an option box with a text field, providing a list box whose contents can be displayed on demand, or edited by typing into the box in its closed state. Any new values typed in by the user are automatically added to the list of options subsequently displayed.

**<spin-box>** A spin box is a text box that will only accept a limited number of input values, themselves making up a loop. A typical example might be the integers between 0 and 10. Spin boxes also incorporate small buttons (up-down controls) that allow the user to change the value by clicking the button in the appropriate direction.

Three controls are also available for displaying more general pieces of information:

**<list-control>** List controls provide an extended list box functionality that let you display a collection of items, each item consisting of an icon and a label. A number of different views are available, allowing you to view the items in different ways.

**<tree-control>** Tree controls are a special list control that displays a set of objects in an indented outline based on the logical hierarchical relationship between the objects.

**<table-control>**

These allow you to display information as a table, with information divided into a number of column headings.

Since all the subclasses of **<collection-gadget>** are themselves value gadgets, each one supports the **value** protocol, as described in Section 8.1 on page 445.

## 8.5 Button gadgets

Broadly speaking, buttons are gadgets whose value can be changed, or for which some user-defined functionality can be invoked, by clicking on the gadget with the pointer device. Buttons encompass obvious controls such as push buttons, radio buttons, and check boxes, and, less obviously, menu items.

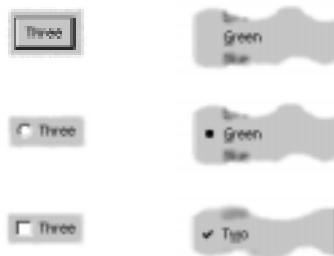


Figure 8.4 A selection of button and equivalent menu buttons

## 8.6 Text gadgets

A text gadget is a gadget into which you can type textual information. There are three different classes of text gadget available in DUIM, each of which is a subclass of the **<text-gadget>** class.

**<text-field>** This is the most basic type of text gadget: the single line edit control. A text field is a gadget into which you can type a single line of text.



**<text-editor>** This is the multiple line edit control, and provides a much richer text editing environment than a text field. This class can be used as the basis for any editing tool, from a simple memo facility to a complex text editor.



**<password-field>**

A password field is really a specialized version of a text field, where the text typed into the gadget is not reflected on the screen. As implied by the name of the class, this type of gadget is most useful when you need to keep the information typed into the field private in some way, such as when implementing a password facility in an application. Depending on the platform you are developing for, any text that you type into a password field is either replaced by asterisks on the screen, or not reflected at all.

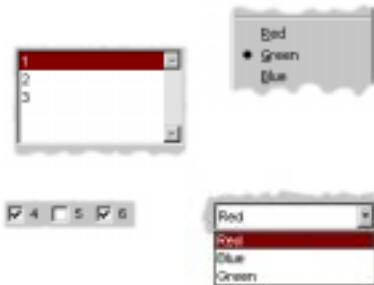


## 8.7 Collection gadgets

A collection gadget is any gadget whose items may themselves form a Dylan collection. Often, a collection gadget is used to group together a number of other gadgets, such as buttons, in such a way that the functionality of those gadgets is connected in some way. For example, a **<radio-box>** is a collection of radio buttons connected in such a way that only one of the buttons can be selected at any time (as is the standard behavior for a group of radio buttons). The items contained in a collection gadget are expressed using the **gadget-items** slot.

Note that collection gadgets are not defined as collections of other gadgets, even though this might be a convenient way to think of them. When defining a collection gadget, you give the `gadget-items` slot a standard Dylan collection. The type of collection gadget you are creating then determines the type of gadget that is contained in the resulting collection gadget.

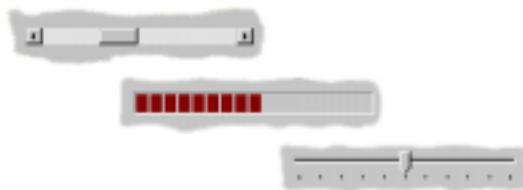
The most simple types of collection gadget mirror the standard buttons and menu buttons available, allowing you to create collections of push buttons, radio buttons, check buttons, and their menu button equivalents. Separators are automatically added to collections of menu buttons so as to delineate them visually from other menu buttons in the same menu.



**Figure 8.5** A variety of simple collection gadgets

## 8.8 Value range gadgets

A value range gadget is any gadget whose value falls within a defined Dylan range.



**Figure 8.6** A variety of value range gadgets

Sliders, scroll bars, and scroll bars are all examples of value range gadgets. Value range gadgets provide immediate visual feedback of the value of the gadget at any time, as shown in Figure 8.6. In the case of sliders and scroll bars, the user can set the `gadget-value` by dragging the appropriate part of the gadget to a new point on the scale. Progress bars are typically used only to provide the user with feedback about the progress of a task.

## 8.9 Page gadgets

A page gadget is used to define the contents of a page in any control that consists of multiple pages. Different classes of page gadget are used for different types of multi-page control. There are three types of page available:

`<tab-control-page>`

These are pages that are used within a tab control.  
Clicking on any tab in a tab control displays a different page of information.

`<property-page>`

These are pages that are displayed in property frames:  
modeless dialog boxes that contain several pages of information displayed as tabbed pages. This class is similar to `<tab-control-page>`, except that its use is

limited to modeless dialog boxes. For more information about property frames, see Chapter 9, “DUIM-Frames Library”.

`<wizard-page>` This type of page is used exclusively in wizard frames, in which the user is guided through a sequence of steps in order to perform a specified operation. For more information about wizard frames, see Chapter 9, “DUIM-Frames Library”.



**Figure 8.7** A tab control page, a property page, and a wizard page

**Note:** The `<wizard-page>` and `<property-page>` classes are actually exposed by the DUIM-Frames library, rather than the DUIM-Gadgets library. See Chapter 9, “DUIM-Frames Library” for full details on this library.

## 8.10 Gadgets that can have children

Most gadgets cannot have any children associated with them; they are leaf elements in the sheet hierarchy. However, a number of specialized gadgets exist which can take children. This section describes those classes.

For all the classes described in this section, the children of any instance of the class are defined using the children: init-keyword. In addition, the children of an instance of any of these classes must themselves be gadgets of some kind. In some cases (menu bars, for instance), the type of gadgets that can be defined as a child is constrained.

### **8.10.1 Menus and menu bars**

You can define a system of menus for a DUIM application by creating a hierarchy of menu bar, menu, and menu button objects. Menu bars can be defined for any application written using DUIM using the `<menu-bar>` class. For most applications, a single menu bar is defined for each window in the application that contains a system of menus. Each menu bar contains a number of menus: the children of the menu bar. Each menu in an application is an instance of the `<menu>` class. The menus of an application can be populated using several different classes of gadget, all of which are subclasses of the `<menu-button>` class.

### **8.10.2 Status bars**

You can add a status bar to a window in a DUIM application by creating an instance of the `<status-bar>` class. A status bar is typically used to provide feedback to the user, and by default shows displays the documentation string for any menu command currently under the mouse cursor. In addition, you can define status bars that display any textual information your application requires, and to this end, status bars can take a number of children.

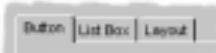


**Figure 8.8** A status bar

In word processing applications, the status bar may also display the current position of the insertion point, and information about the current font family, size, and variation, if appropriate. In an e-mail client application, the status bar may display the number of messages in the current folder. Often, the system time is displayed in the status bar for an application.

### 8.10.3 Tab controls

An instance of the class `<tab-control>` lets you define a sheet that contains several “pages” of information. Each page of information is displayed by clicking on the appropriate tab along the top of the sheet.

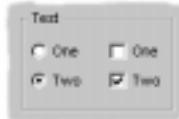


**Figure 8.9** A tab control

This children of a tab control are the pages of information themselves. Each child should be an instance of the `<page>` class. The various types of page available are described in Section 8.9 on page 462.

### 8.10.4 Group boxes

The `<group-box>` class allows you to group together any number of gadgets that are associated to some degree in an interface. A group box creates a purely visual grouping, and does not affect the behavior or interaction between its children in any way. For this reason, there are no constraints on the types of gadget that you can group together; the children of a group box can be any type of gadget.



## 8.11 DUIM-Gadgets Module

This section contains a complete reference of all the interfaces that are exported from the `duim-gadgets` module.

<b>&lt;action-gadget&gt;</b>		<i>Open abstract class</i>
Summary	The protocol class for gadgets that have action callbacks.	
Superclasses	<code>&lt;gadget&gt;</code>	
Init-keywords	<code>activate-callback:</code>	An instance of type <code>false-or(&lt;function&gt;)</code> . Default value: <code>#f</code> .
Description	The class used by gadgets that have an action callback that allows some type of action to be performed, such as a push button. Action gadgets can only be activated when they are enabled.	
Operations	<code>gadget-activate-callback</code> <code>gadget-activate-callback-setter</code>	
See also	<code>&lt;gadget&gt;</code> , page 485	

<b>activate-gadget</b>		<i>Generic function</i>
Summary	Activates the specified gadget.	
Signature	<code>activate-gadget <i>gadget</i> =&gt; ()</code>	
Arguments	<code>gadget</code>	An instance of type <code>&lt;gadget&gt;</code> .
Values	None	
Description	Activates gadget by calling the activate callback. For example, in the case of a button, calling this generic function would be as if the user had pressed the button.	

	<i>Generic function</i>
Summary	Adds a column to the specified table.
Signature	<code>add-column <i>table heading generator index</i> =&gt; ()</code>
Arguments	<p><i>table</i> An instance of type <code>&lt;table-control&gt;</code>.</p> <p><i>heading</i> An instance of type <code>type-union(&lt;string&gt;, &lt;label&gt;)</code>.</p> <p><i>generator</i> An instance of type <code>&lt;function&gt;</code>.</p> <p><i>index</i> An instance of type <code>&lt;integer&gt;</code>.</p>
Values	None
Description	Adds a column <i>table</i> , with a table heading given by <i>heading</i> . The contents of the column are generated by calling the <i>generator</i> function on the item for each row of <i>table</i> . The <i>index</i> specifies where in the column order the new column should be added.
See also	<a href="#">remove-column</a> , page 559

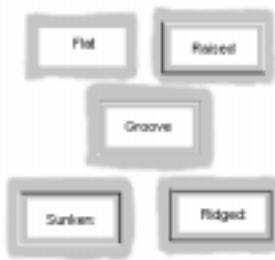
	<i>Generic function</i>
Summary	Adds an item to the specified list or table control.
Signature	<code>add-item <i>list-or-table item #key after</i> =&gt; <i>item</i></code>
Arguments	<p><i>list-or-table</i> An instance of <code>type-union(&lt;list-control&gt;, &lt;table-control&gt;)</code>.</p> <p><i>item</i> An instance of type <code>type-union(&lt;list-item&gt;, &lt;table-item&gt;)</code>.</p> <p><i>after</i> An instance of type <code>type-union(&lt;list-item&gt;, &lt;table-item&gt;)</code>.</p>

Values	<i>item</i>	An instance of type <code>type-union(&lt;list-item&gt;, &lt;table-item&gt;)</code> .
Description	<p>Adds an <i>item</i> to the specified <i>list-or-table</i>. The new item is created via a call to <code>make-item</code>.</p> <p>The <i>after</i> argument indicates which item to place the new item after.</p>	
See also	<p><code>find-item</code>, page 484</p> <p><code>&lt;list-control&gt;</code>, page 531</p> <p><code>&lt;list-item&gt;</code>, page 537</p> <p><code>make-item</code>, page 538</p> <p><code>remove-item</code>, page 560</p> <p><code>&lt;table-control&gt;</code>, page 586</p> <p><code>&lt;table-item&gt;</code>, page 592</p>	

<b>add-node</b>		<i>Generic function</i>
Summary		Adds node to the specified tree control.
Signature		<code>add-node tree parent node #key after setting-roots? =&gt; node</code>
Arguments	<code>tree</code> <code>parent</code> <code>node</code> <code>after</code> <code>setting-roots?</code>	An instance of <code>&lt;tree-control&gt;</code> . An instance of <code>&lt;tree-control&gt;</code> . An instance of type <code>&lt;tree-node&gt;</code> . An instance of type <code>&lt;tree-node&gt;</code> . An instance of type <code>&lt;boolean&gt;</code> .
Values	<code>node</code>	An instance of type <code>&lt;tree-node&gt;</code> .

Description	Adds a <i>node</i> to the specified <i>tree</i> with the specified <i>parent</i> . The new item is created via a call to <code>make-node</code> .
	The <i>after</i> argument indicates which node to place the new node after. If <i>setting-roots?</i> is true, then the new node is added at the root of <i>tree</i> .
See also	<code>find-node</code> , page 485 <code>make-node</code> , page 540 <code>remove-node</code> , page 560 <code>&lt;tree-control&gt;</code> , page 598

<b>&lt;border&gt;</b> <i>Open abstract instantiable class</i>	
Summary	The class of bordering gadgets.
Superclasses	<code>&lt;gadget&gt;</code> <code>&lt;single-child-composite-pane&gt;</code>
Init-keywords	<p><b>thickness:</b> An instance of type <code>&lt;integer&gt;</code>. Default value: 1.</p> <p><b>type:</b> An instance of type <code>one-of(#f, #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output")</code>. Default value: <code>#f</code>.</p>
Description	<p>The base class of gadgets that provide borders to their children.</p> <p>The thickness of the border is specified by the <code>thickness:</code> init-keyword, and is given in pixels.</p> <p>The <code>type:</code> init-keyword represents the kind of border to be created. Borders may appear raised from the area they surround, or lowered with respect to it. Alternatively, a border may be displayed as a thin ridge or groove. Input and output borders represent “logical” borders.</p>



**Figure 8.10** Different types of border

Borders are usually created using the `with-border` macro, rather than by making direct instances of this class.

Operations	None.
See also	<code>&lt;group-box&gt;</code> , page 526 <code>with-border</code> , page 613

<button>

### ***Open abstract instantiable class***

Summary	The class of all button gadgets.
Superclasses	<value-gadget>
Init-keywords	<b>accelerator:</b> An instance of type <b>false-or(&lt;gesture&gt;)</b> . Default value: #f.  <b>mnemonic:</b> An instance of type <b>false-or(&lt;character&gt;)</b> . Default value: #f.
Description	The class of all button gadgets.  The <b>accelerator:</b> init-keyword is used to specify a keyboard accelerator for the button. This is a key press that gives the user a method for activating the button using a short key sequence rather than by clicking the button itself. Keyboard

accelerators usually combine the CONTROL and possibly SHIFT keys with an alphanumeric character.

When choosing accelerators, you should be aware of style guidelines that might be applicable for the operating system you are developing for. For example, a common accelerator for the command **File > Open** in Windows is CTRL+O.

Keyboard accelerators are mostly used in menu buttons, though they can be applied to other forms of button as well.

The **mnemonic**: init-keyword is used to specify a keyboard mnemonic for the button. This is a key press that involves pressing the ALT key followed by a number of alphanumeric keys.

Note that the choice of keys is more restrictive than for keyboard accelerators. They are determined in part by the names of button itself (and, in the case of menu buttons, the menu that contains it), as well as by any appropriate style guidelines. For example, a common mnemonic for the **File > Open** command is ALT, F, O.

Mnemonics have the advantage that the letters forming the mnemonic are automatically underlined in the button label on the screen (and, for menu buttons, the menu itself). This means that they do not have to be remembered. In addition, when the user makes use of a mnemonic in a menu, the menu itself is displayed on screen, as if the command had been chosen using the mouse. This does not happen if the keyboard accelerator is used.

Buttons are intrinsically “non-stretchy” gadgets. That is, the width and height of a button is generally calculated on the basis of the button’s label, and the button will be sized so that it fits the label accordingly. Sometimes, however, you want a button to occupy all the available space that is given it, however large that space may be. To force a button to use all the

available width or height, specify `max-width: $fill` or `max-height: $fill` accordingly in the button definition. See the second example below to see how this is done.

Operations	<code>dialog-apply-button-setter dialog-back-button-setter dialog-cancel-button-setter dialog-exit-button-setter dialog-help-button-setter dialog-next-button-setter dialog-apply-button-setter</code>
Example	<pre>contain   (make(&lt;button&gt;, label: "Hello",         activate-callback:           method (gadget)             notify-user               (format-to-string                 ("Pressed button %=", gadget),                 owner: gadget)             end)); </pre>

The following example creates a column layout that contains two elements.

- The first is a row layout that itself contains two buttons with short labels.
- The second is a button with a long label.

The use of `equalize-widths?:` in the call to `vertically` ensures that these two elements have the same width.

The interesting part of this example is in the use of `max-width: $fill` in the definition of the buttons with shorter labels. If this was not used, then each button would be sized such that it just fit its own label, and there would be empty space in the row layout. However, using `max-width: $fill` ensures that each button is made as large as possible, so as to fit the entire width of the row layout.

```

vertically (equalize-widths?: #t)
    horizontally ()
        make(<button>, label: "Red", max-width: $fill);
        make(<button>, label: "Ultraviolet",
              max-width: $fill);
    end;
    make(<button>,
          label:
              "A button with a really really long label");
end

```

See also      `<button-box>`, page 473  
               `<check-button>`, page 476  
               `$fill`, page 410  
               `gadget-accelerator`, page 488  
               `<menu-button>`, page 544  
               `<radio-button>`, page 557  
               `<space-requirement>`, page 427

**<button-box>***Open abstract instantiable class*

Summary      A class that groups buttons.

Superclasses    `<collection-gadget>` `<multiple-child-composite-pane>`

Init-keywords    **rows:**        An instance of type `false-or(<integer>)`.  
                   **columns:**     An instance of type `false-or(<integer>)`.  
                   **orientation:**   An instance of type `one-of(#"horizontal",`  
                           `#"vertical")`. Default value:  
                           `#"horizontal"`.  
                   **layout-class:** An instance of type `subclass(<layout>)`.  
                           Default value: `<column-layout>` or `<row-layout>`, depending on orientation.

**child:** An instance of type **false-or(<sheet>)**.  
Default value: #f.

Description The class of grouped buttons; the superclass of **<check-box>** and **<radio-box>**.

The **rows:** and **columns:** init-keywords allow you to specify how many rows or columns should be used to lay out the buttons. In addition, you can set the orientation of the button box by specifying the **orientation:** init-keyword.

An instance of the class that is specified by **layout-class:** is used to parent the buttons that are created, and any extra arguments that are specified, such as **x-alignment:** and **x-spacing:**, are passed along to this layout.

You can use the **child:** init-keyword to specify a sheet hierarchy to be used in place of a list of items. Under normal circumstances, the items defined for any button box are realized in terms of their “natural” gadget class. For example, if you create a radio button box, DUIM creates a radio button for each item that you specify. By using the **child:** init-keyword, you can define sheet hierarchies that override these “natural” gadget classes, letting you specify more complex arrangements of gadgets: in this way, you could create a check button box where each check button is itself surrounded by a group box. For an example of the use of the **child:** init-keyword, look at the initial dialog box that is displayed when you first start the Dylan environment. In this dialog, a number of radio buttons are presented, each delineated by its own group box. In fact, this dialog is implemented as a radio button box in which the **child:** init-keyword has been used rather than the **items:** init-keyword.

If you use **child:**, then the **gadget-value** returned by the gadget is the **gadget-id** of the selected button. Contrast this with **items:**, where the selected item is returned as the **gadget-value**.

Operations      None.

Examples      

```
contain(make(<button-box>,
              selection-mode: #'multiple",
              items: range(from: 0, to: 20)));
```

The following examples illustrate the use of some of the init-keywords described. They each create an instance of a subclass of `<button-box>`. Note that the `selection-mode:` init-keyword may be used instead, rather than creating a direct instance of one of the subclasses.

```
contain(make(<check-box>, items: range(from: 1, to: 9),
             columns: 3));

contain(make(<radio-box>, items: #("Yes", "No"),
             orientation: #'vertical');

contain(make(<check-box>, items: #(1, 2, 3, 4),
             layout-class: <table-layout>
             rows: 2));
```

See also      `<check-box>`, page 475

`<push-box>`, page 552

`<radio-box>`, page 556

## `<check-box>`

*Open abstract instantiable class*

Summary      The class of check boxes, or groups of check buttons.

Superclasses    `<button-box> <action-gadget>`

Init-keywords    None.

Description      The instantiable class that implements an abstract check box, that is, a gadget that constrains a number of toggle buttons, zero or more of which may be selected at any one time.



The value of a check box is a sequence of all the currently selected items in the check box.

Operations	None.
Examples	<pre>contain(make(&lt;check-box&gt;, items: #(1, 2, 3, 4, 5)); contain(make(&lt;check-box&gt;, items: range(from: 1, to: 9),             columns: 3)); contain(make(&lt;check-box&gt;, items: #(1, 2, 3, 4),             layout-class: &lt;table-layout&gt;             rows: 2));</pre>
See also	<p>&lt;<a href="#">group-box</a>&gt;, page 526</p> <p>&lt;<a href="#">push-box</a>&gt;, page 552</p> <p>&lt;<a href="#">radio-box</a>&gt;, page 556</p>

## <check-button>

*Open abstract instantiable class*

Summary	The class of check buttons.
Superclasses	< <a href="#">button</a> > < <a href="#">action-gadget</a> >
Init-keywords	None.
Description	<p>The class of check buttons. The value of a check button is either <code>#t</code> or <code>#f</code>, depending whether or not it is currently selected.</p>  <p>Internally, this class maps into the check box Windows control.</p>
Operations	None.
Example	<code>contain(make(&lt;check-button&gt;, label: "Check button"));</code>
See also	< <a href="#">check-menu-button</a> >, page 478

[`<push-button>`](#), page 553

[`<radio-button>`](#), page 557

## `<check-menu-box>`

*Open abstract instantiable class*

**Summary** The class of groups of check buttons displayed in a menu.

**Superclasses** `<menu-box>` `<action-gadget>`

**Init-keywords** None.

**Description** The class of groups of check buttons displayed in a menu.

Internally, this class maps into the menu Windows control.



**Operations** None.

**Example** The following example creates a menu that shows an example of a check menu box.

```
contain(make(<menu>,
            label: "Hello...",
            children: vector
                (make(<radio-menu-box>,
                      items:
                        #("You", "All",
                           "Everyone")),
             )));
```

**See also** [`<menu-box>`](#), page 543

[`<push-menu-box>`](#), page 554

[`<radio-menu-box>`](#), page 557

**<check-menu-button>***Open abstract instantiable class*

**Summary**      The class of check buttons that can be displayed in a menu.

**Superclasses**    <menu-button>

**Init-keywords**    None.

**Description**      The class of check buttons that can be displayed in a menu. The values of a menu button is either #t or #f.



Internally, this class maps into the menu item Windows control.

**Operations**      None.

**Example**

```
contain
  (make(<check-menu-button>,
        label: "Menu button",
        activate-callback:
          method (gadget)
            notify-user(format-to-string
              ("Toggled button %=", gadget)) end));
```

**See also**        <check-button>, page 476

                  <radio-menu-button>, page 558

**<collection-gadget>***Open abstract class*

**Summary**      The class of all gadgets that contain collections.

**Superclasses**    <value-gadget>

**Init-keywords**    **items:**      An instance of type <sequence>. Default value: #[ ].

**label-key:**     An instance of type <function>.

**value-key:** An instance of type <function>. Default value: `identity`.

**test:** An instance of type <function>. Default value: `==`.

**selection:** An instance of type `limited(<sequence>, of: <integer>)`. Default value: `#[]`.

**selection-mode:**  
An instance of type `one-of(#"single", #"multiple", #"none")`. Default value: `#"single"`.

**key-press-callback:**  
An instance of type `false-or(<command>, <function>)`.

Description	<p>The class of all gadgets that can contain collections.</p> <p>The <code>items</code>: init-keyword is used to specify the collection of items that the collection gadget contains.</p> <p>The <code>label-key</code>: and <code>value-key</code>: init-keywords are functions that are used to calculate the labels and the value of the gadget respectively.</p> <p>The value of a collection gadget is determined by calling the value key of the gadget on each selected item in the gadget. The “printed representation” of a collection gadget is determined by calling the label key of the gadget on each item.</p> <p>By default, the label key returns the numeric label of the gadget items (for example, the buttons in a button box would be labeled 1, 2, 3, and so on). In general, the label key can be trusted to “do the right thing” by default.</p> <p>By default, the value key returns the collection gadget itself.</p>
-------------	--

Note also that the `gadget-value` method for collection gadgets is different for single and multiple selection gadgets. For single selection, the item that is selected is returned. For multiple selection, a sequence of the selected items is returned.

The `test:` init-keyword is the function used to test whether two items of the collection are considered identical.

The `selection:` init-keyword is available only to those subclasses of `<collection-gadget>` that contain items that may be selected. The selection is a collection containing the selected keys from the items collection.

Subclasses of `<collection-gadget>` that can have selections are: `<list-box>`, `<option-box>`, `<list-control>`, `<tree-control>`, `<table-control>`, `<radio-box>`, `<check-box>`, `<check-menu-box>`, `<radio-menu-box>`, `<combo-box>`.

The `key-press-callback:` init-keyword lets you specify a key-press callback. This type of callback is invoked whenever a key on the keyboard is pressed while the gadget has focus. It applies only to graph controls, list controls, tab controls, and table controls. See `gadget-key-press-callback`, page 502, for a fuller description of key-press callbacks.

Operations	<code>gadget-items</code> <code>gadget-items-setter</code> <code>gadget-key-press-callback</code> <code>gadget-key-press-callback-setter</code> <code>gadget-label-key</code> <code>gadget-selection</code> <code>gadget-selection-mode</code> <code>gadget-selection-setter</code> <code>gadget-test</code> <code>gadget-value-key</code>
See also	<a href="#">&lt;button-box&gt;</a> , page 473 <a href="#">&lt;check-box&gt;</a> , page 475 <a href="#">&lt;check-menu-box&gt;</a> , page 477 <a href="#">&lt;combo-box&gt;</a> , page 481 <a href="#">&lt;list-box&gt;</a> , page 529 <a href="#">&lt;list-control&gt;</a> , page 531

**<option-box>**, page 549  
**<radio-box>**, page 556  
**<radio-menu-box>**, page 557  
**<table-control>**, page 586  
**<tree-control>**, page 598

## **<combo-box>**

*Open abstract instantiable class*

**Summary** The class of combo boxes, which combine options boxes with text fields.

**Superclasses** `<collection-gadget> <action-gadget> <text-gadget>`

**Init-keywords** **borders:** An instance of type `one-of(#f, #"none", #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output")`. Default value: `#f`.

**scroll-bars:** An instance of type `one-of(#f, #"none", #"horizontal", #"vertical", #"both", #"dynamic")`. Default value: `#"both"`.

**Description** The class of combo boxes. Combo boxes are similar to option boxes, except that the text field is editable, so that new values can be specified in addition to those already provided in the drop-down list. Users may either choose an existing option from the list, or type in their own.



It is common for additional items typed by the user to be added to the list of options available. A combo box is often used to specify text in a Find dialog box, for example, and any previous search terms can be recalled by choosing them from the list. If you wish to provide this functionality, then

you can do so using a combination of `add-item` and `find-item`, to search for the presence of an item and add it if it does not already exist.

The `borders:` init-keyword lets you specify a border around the combo box. If specified, a border of the appropriate type is drawn around the gadget.

The `scroll-bars:` init-keyword lets you specify the scroll bar behavior for the gadget.

Internally, this class maps into the Windows combo box control.

Operations      None.

Example      `contain(make(<combo-box>, value-type: <integer>  
                  items: range(from: 1 to: 5))),`

See also      `<option-box>`, page 549  
`<text-field>`, page 593

## contract-node

## *Generic function*

Summary      Contracts the specified node in a tree control.

Signature      `contract-node tree-control node => ()`

Arguments      `tree-control`      An instance of `<tree-control>`.  
`node`          An instance of type `<tree-node>`.

Values      None

Description      Contracts the specified `node` in `tree-control`, thereby hiding any children of the node that were displayed.

See also      `expand-node`, page 483

**display-menu** *Generic function*

Summary	Displays the specified menu.	
Signature	<code>display-menu <i>menu</i> #key <i>x y</i> =&gt; ()</code>	
Arguments	<i>menu</i>	An instance of type <code>&lt;menu&gt;</code> .
	<i>x</i>	An instance of type <code>false-or(&lt;integer&gt;)</code> . Default value: #f.
	<i>y</i>	An instance of type <code>false-or(&lt;integer&gt;)</code> . Default value: #f.
Values	None	
Description	<p>Displays the specified menu, optionally at a precise position on the screen, specified by <i>x</i> and <i>y</i>, where <i>x</i> and <i>y</i> are both relative to the owner of the menu.</p> <p>The function returns when the menu has been popped down again.</p>	
See also	<code>&lt;menu&gt;</code> , page 541	

**expand-node** *Generic function*

Summary	Expands the specified node in a tree control.	
Signature	<code>expand-node <i>tree-control</i> <i>node</i> #key <i>sort-function</i> =&gt; ()</code>	
Arguments	<i>tree-control</i>	An instance of <code>&lt;tree-control&gt;</code> .
	<i>node</i>	An instance of type <code>&lt;tree-node&gt;</code> .
Values	None	

Description	Expands the specified node in a <i>tree-control</i> , thereby displaying any children that the node has.  If no children have been explicitly added to the node before it is expanded, they are generated by calling the tree's children generating function on the node.
See also	<a href="#">contract-node</a> , page 482 <a href="#">tree-control-children-generator</a> , page 602

## **find-item** *Generic function*

Summary	Finds an item in a list control or a table control.				
Signature	<b>find-item</b> <i>list-or-table object #key =&gt; found-item</i>				
Arguments	<table> <tr> <td><i>list-or-table</i></td><td>An instance of <i>type-union</i>(<i>&lt;list-control&gt;</i>, <i>&lt;table-control&gt;</i>).</td></tr> <tr> <td><i>object</i></td><td>An instance of type <i>&lt;object&gt;</i>.</td></tr> </table>	<i>list-or-table</i>	An instance of <i>type-union</i> ( <i>&lt;list-control&gt;</i> , <i>&lt;table-control&gt;</i> ).	<i>object</i>	An instance of type <i>&lt;object&gt;</i> .
<i>list-or-table</i>	An instance of <i>type-union</i> ( <i>&lt;list-control&gt;</i> , <i>&lt;table-control&gt;</i> ).				
<i>object</i>	An instance of type <i>&lt;object&gt;</i> .				
Values	<i>found-item</i> An instance of type <i>type-union(&lt;list-item&gt;, &lt;table-item&gt;, #f)</i> .				
Description	Finds the item in a list control or a table control that corresponds to <i>object</i> .				
See also	<a href="#">add-item</a> , page 467 <a href="#">&lt;list-control&gt;</a> , page 531 <a href="#">&lt;list-item&gt;</a> , page 537 <a href="#">make-item</a> , page 538 <a href="#">remove-item</a> , page 560 <a href="#">&lt;table-control&gt;</a> , page 586 <a href="#">&lt;table-item&gt;</a> , page 592				

**find-node** *Generic function*

Summary	Finds a node in a tree control.						
Signature	<code>find-item tree object #key parent-node =&gt; found-item</code>						
Arguments	<table> <tr> <td><i>tree</i></td><td>An instance of &lt;tree-control&gt;.</td></tr> <tr> <td><i>object</i></td><td>An instance of &lt;object&gt;.</td></tr> <tr> <td><i>parent-node</i></td><td>An instance of type &lt;tree-node&gt;.</td></tr> </table>	<i>tree</i>	An instance of <tree-control>.	<i>object</i>	An instance of <object>.	<i>parent-node</i>	An instance of type <tree-node>.
<i>tree</i>	An instance of <tree-control>.						
<i>object</i>	An instance of <object>.						
<i>parent-node</i>	An instance of type <tree-node>.						
Values	<i>found-item</i> An instance of type <tree-node>.						
Description	Finds the item in a tree control that corresponds to <i>object</i> .						
See also	<a href="#">add-node</a> , page 468 <a href="#">make-node</a> , page 540 <a href="#">remove-node</a> , page 560 <a href="#">&lt;tree-control&gt;</a> , page 598						

**<gadget>** *Open abstract class*

Summary	The protocol class of all gadgets.								
Superclasses	<object>								
Init-keywords	<table> <tr> <td><b>id:</b></td><td>An instance of type <code>false-or(&lt;object&gt;)</code>. Default value: #f.</td></tr> <tr> <td><b>client:</b></td><td>An instance of type <code>false-or(&lt;object&gt;)</code>. Default value: #f.</td></tr> <tr> <td><b>label:</b></td><td>An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code>. Required.</td></tr> <tr> <td><b>documentation:</b></td><td>An instance of type <code>false-or(&lt;string&gt;)</code>. Default value: #f.</td></tr> </table>	<b>id:</b>	An instance of type <code>false-or(&lt;object&gt;)</code> . Default value: #f.	<b>client:</b>	An instance of type <code>false-or(&lt;object&gt;)</code> . Default value: #f.	<b>label:</b>	An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code> . Required.	<b>documentation:</b>	An instance of type <code>false-or(&lt;string&gt;)</code> . Default value: #f.
<b>id:</b>	An instance of type <code>false-or(&lt;object&gt;)</code> . Default value: #f.								
<b>client:</b>	An instance of type <code>false-or(&lt;object&gt;)</code> . Default value: #f.								
<b>label:</b>	An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code> . Required.								
<b>documentation:</b>	An instance of type <code>false-or(&lt;string&gt;)</code> . Default value: #f.								

	<b>enabled?:</b> An instance of type <boolean>. Default value: #t.
	<b>read-only?:</b> An instance of type <boolean>. Default value: #f.
Description	<p>The class of all gadgets. You should not create a direct instance of this class.</p> <p>The <b>id:</b> init-keyword lets you specify a unique identifier for the action gadget. This is a useful way of identifying gadgets, and provides you with an additional way of controlling execution of your code, allowing you to create simple branching statements such as:</p> <pre>select (gadget-id)     #"ok" =&gt; do-okay();     #"cancel" =&gt; do-cancel(); end select;</pre> <p>Note, however, that specifying <b>id:</b> is not generally necessary. The <b>id:</b> init-keyword is useful in the case of tab controls, where it is returned by <b>gadget-value</b>.</p> <p>Every gadget has a <b>client:</b> that is specified when the gadget is created. Typically, <b>client:</b> is a frame or a composite sheet.</p> <p>The <b>label:</b> init-keyword lets you assign a label to any gadget. A label may be any string, or an image of an appropriate size (usually a small icon).</p> <p>The <b>documentation:</b> init-keyword is used to provide a short piece of online help for the gadget. Any documentation supplied for a gadget may be used in a tooltip or a status bar. For example, moving the mouse over a menu command may display the supplied documentation for that command in the status bar of your application, or moving the mouse over any of the buttons in a tool bar may display a tooltip (a piece of pop-up text) that contains the supplied documentation.</p>

If `enabled?:` is true, then the gadget is enabled; that is, the user can interact with the gadget in an appropriate way. If the gadget is not enabled, then the user cannot interact with it. For example, if a push button is not enabled, it cannot be clicked, or if a check box is not enabled, its setting cannot be switched on or off. Gadgets that are not enabled are generally grayed out on the screen.

If `read-only?:` is true, then the user cannot alter any of the values displayed in the gadget; this typically applies to text gadgets. Note that this is not the same as disabling the gadget — if a gadget is set to read-only, it is not grayed out, and the user may still interact with it: only the values cannot be changed.

Operations	<code>activate-gadget</code> <code>choose-from-dialog</code> <code>gadget-accelerator</code> <code>gadget-accelerator-setter</code> <code>gadget-client</code> <code>gadget-client-setter</code> <code>gadget-command</code> <code>gadget-command-setter</code> <code>gadget-default?</code> <code>gadget-default?-setter</code> <code>gadget-documentation</code> <code>gadget-documentation-setter</code> <code>gadget-value-changing</code> <code>callback</code> <code>gadget-value-changing-callback-setter</code> <code>gadget-enabled?</code> <code>gadget-enabled?-setter</code> <code>gadget-id</code> <code>gadget-id-setter</code> <code>gadget-label</code> <code>gadget-label-setter</code> <code>gadget-mnemonic</code> <code>gadget-mnemonic-setter</code> <code>gadget-orientation</code> <code>gadget-popup-menu-callback</code> <code>gadget-popup-menu-callback-setter</code> <code>gadget-read-only?</code> <code>gadget-scrolling-horizontally?</code> <code>gadget-scrolling-vertically?</code> <code>update-gadget</code>
See also	<code>&lt;action-gadget&gt;</code> , page 466 <code>&lt;border&gt;</code> , page 469 <code>gadget-value</code> , page 517 <code>&lt;group-box&gt;</code> , page 526 <code>&lt;label&gt;</code> , page 528

<menu>, page 541  
<page>, page 550  
<separator>, page 564  
<spacing>, page 569  
<tool-bar>, page 597  
<value-gadget>, page 608  
<viewport>, page 610

**gadget?** *Generic function*

Summary	Returns true if the specified object is a gadget.
Signature	<code>gadget? object =&gt; gadget?</code>
Arguments	<code>object</code> An instance of type <object>.
Values	<code>gadget?</code> An instance of type <boolean>.
Description	Returns true if <i>object</i> is a gadget.
Example	<pre>*gadget* := contain(make                       (&lt;radio-menu-box&gt;,                        items: range(from: 0, to: 20))); gadget?(*gadget*);</pre>
See also	<gadget>, page 485

**gadget-accelerator** *Generic function*

Summary	Returns the keyboard accelerator of the specified gadget.
Signature	<code>gadget-accelerator gadget =&gt; accelerator</code>

Arguments	<i>gadget</i>	An instance of type < <code>gadget</code> >.
Values	<i>accelerator</i>	An instance of type < <code>gesture</code> >.
Description		Returns the keyboard accelerator of the specified gadget. An accelerator is a keyboard gesture that activates a gadget (that is, it invokes the activate callback for the gadget) without needing to use the mouse.  Accelerators are of most use with button gadgets, and in particular menu button gadgets.
See also		<code>&lt;button&gt;</code> , page 470 <code>gadget-accelerator-setter</code> , page 489 <code>&lt;gesture&gt;</code> , page 247 <code>&lt;menu-button&gt;</code> , page 544

<b>gadget-accelerator-setter</b>		<i>Generic function</i>
Summary		Sets the keyboard accelerator of the specified gadget.
Signature		<code>gadget-accelerator-setter accelerator gadget =&gt; accelerator</code>
Arguments	<i>accelerator</i>	An instance of type < <code>gesture</code> >.
	<i>gadget</i>	An instance of type < <code>gadget</code> >.
Values	<i>accelerator</i>	An instance of type < <code>gesture</code> >.
Description		Sets the keyboard accelerator of the specified gadget. An accelerator is a keyboard gesture that invokes the activate callback of a gadget without needing to use the mouse.  Accelerators are of most use with button gadgets, and in particular menu button gadgets.

See also      **<button>**, page 470  
                **gadget-accelerator**, page 488  
                **<gesture>**, page 247  
                **<menu-button>**, page 544

## **gadget-activate-callback** *Generic function*

Summary	Returns the activate callback of the specified gadget.
Signature	<code>gadget-activate-callback <i>gadget</i> =&gt; <i>activate-callback</i></code>
Arguments	<i>gadget</i> An instance of type <code>&lt;action-gadget&gt;</code> .
Values	<i>activate-callback</i> An instance of type <code>false-or(&lt;function&gt;)</code> .
Description	Returns the function that will be called when <i>gadget</i> is activated. This function will be invoked with one argument, the gadget itself.  When this function returns <code>#f</code> , this indicates that there is no activate callback for the gadget.
See also	<code>gadget-activate-callback-setter</code> , page 490

gadget-activate-callback-setter	Generic function				
Summary	Sets the activate callback for the specified gadget.				
Signature	<code>gadget-activate-callback-setter activate-callback gadget =&gt; activate-callback</code>				
Arguments	<table border="0"><tr><td data-bbox="404 1306 588 1325">activate-callback</td><td data-bbox="588 1306 1081 1325">An instance of type <code>false-or(&lt;function&gt;)</code>.</td></tr><tr><td data-bbox="404 1325 588 1346">gadget</td><td data-bbox="588 1325 1081 1346">An instance of type <code>&lt;action-gadget&gt;</code>.</td></tr></table>	activate-callback	An instance of type <code>false-or(&lt;function&gt;)</code> .	gadget	An instance of type <code>&lt;action-gadget&gt;</code> .
activate-callback	An instance of type <code>false-or(&lt;function&gt;)</code> .				
gadget	An instance of type <code>&lt;action-gadget&gt;</code> .				

Values	<code>activate-callback</code> An instance of type <code>false-or(&lt;function&gt;)</code> .
Description	Sets the activate callback for <i>gadget</i> to <i>function</i> .
See also	<code>gadget-activate-callback</code> , page 490

## **gadget-client** *Generic function*

Summary	Returns the client of the specified gadget.
Signature	<code>gadget-client gadget =&gt; client</code>
Arguments	<code>gadget</code> An instance of type <code>&lt;gadget&gt;</code> .
Values	<code>client</code> An instance of type <code>&lt;object&gt;</code> .
Description	<p>Returns the client of <i>gadget</i>. The client is the gadget or frame that <i>gadget</i> should look to for callback information.</p> <p>In any sheet hierarchy, the client is usually the immediate parent of <i>gadget</i>. This often means that the client is a frame, but it can also be another gadget. In the majority of cases, you need not be concerned with the client of a gadget. However, rather like the <code>gadget-id</code>, you are free to assign your own client to a given gadget whenever it is necessary for your code.</p> <p>In less obvious cases, the client may not be the immediate parent: for example, in the case of a radio box, the client of each button in the radio box is the radio box itself. At the implementation level, the radio box is not the immediate parent of the buttons that it contains, since there is an intervening layout object that arranges the buttons within the box. See <code>&lt;action-gadget&gt;</code>, page 466, for more details.</p> <p>Gadget clients enable you to pass messages between the gadget and its client when a callback is received.</p>

See also [gadget-client-setter](#), page 492

## **gadget-client-setter** *Generic function*

Summary Sets the client of the specified gadget.

Signature `gadget-client-setter client gadget => client`

Arguments `client` An instance of type `<object>`.

`gadget` An instance of type `<gadget>`.

Values `client` An instance of type `<object>`.

Description Sets the *client* of the specified *gadget*.

The client is often a frame, but it could be another gadget (for example, in the case of a push button that is contained in a radio box, the client of the button could be the radio box). See [<action-gadget>](#), page 466, for more details.

Gadget clients enable you to pass messages between the gadget and its client when a callback is received.

See also [gadget-client](#), page 491

## **gadget-command** *Generic function*

Summary Returns the command associated with the specified gadget.

Signature `gadget-command gadget => command`

Arguments `gadget` An instance of type `<gadget>`.

Values `command` An instance of type `false-or(<command>)`.

Description	Returns the command associated with <i>gadget</i> .  A command is typically associated with a gadget if that gadget has been created by using a command table. For example, the command associated with a menu button would represent the callback that is invoked when the user chooses the relevant menu command.
See also	<a href="#">gadget-command-setter</a> , page 493

**gadget-command-setter** *Generic function*

Summary	Sets the command of the specified gadget.	
Signature	<code>gadget-command-setter command gadget =&gt; command</code>	
Arguments	<i>command</i>	An instance of type <code>false-or(&lt;command&gt;)</code> .
	<i>gadget</i>	An instance of type <code>&lt;gadget&gt;</code> .
Values	<i>command</i>	An instance of type <code>false-or(&lt;command&gt;)</code> .
Description	Sets the command of the specified <i>gadget</i> .  A command is typically associated with a gadget if that gadget has been created by using a command table. For example, the command associated with a menu button would represent the callback that is invoked when the user chooses the relevant menu command.	
See also	<a href="#">gadget-command</a> , page 492	

**gadget-default?** *Generic function*

Summary	Returns true if the specified gadget is the default gadget in a frame.
---------	--

Signature	<code>gadget-default? gadget =&gt; default?</code>	
Arguments	<code>gadget</code>	An instance of type <code>&lt;gadget&gt;</code> .
Values	<code>default?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	<p>Returns true if the specified gadget is the default gadget for the frame it is part of.</p> <p>It is generally useful to set a default gadget in a frame, or a default menu if there is no suitable gadget.</p> <p>When a default gadget is specified, using the default keyboard gesture in the frame invokes the activate callback for the default gadget. The default gesture is usually pressing the RETURN button.</p>	
See also	<code>gadget-default?-setter</code> , page 494	

	<i>Generic function</i>				
<b>gadget-default?-setter</b>					
Summary	Toggles whether the specified button is the default for the current frame.				
Signature	<code>gadget-default?-setter default? button =&gt; default?</code>				
Arguments	<table> <tr> <td><code>default?</code></td> <td>An instance of type <code>&lt;boolean&gt;</code>.</td> </tr> <tr> <td><code>button</code></td> <td>An instance of type <code>&lt;button&gt;</code>.</td> </tr> </table>	<code>default?</code>	An instance of type <code>&lt;boolean&gt;</code> .	<code>button</code>	An instance of type <code>&lt;button&gt;</code> .
<code>default?</code>	An instance of type <code>&lt;boolean&gt;</code> .				
<code>button</code>	An instance of type <code>&lt;button&gt;</code> .				
Values	<code>default?</code> An instance of type <code>&lt;boolean&gt;</code> .				
Description	If <code>default?</code> is true, <code>button</code> becomes the default gadget for the current frame. If <code>default?</code> is #f, <code>button</code> is not the default gadget for the current frame, regardless of any previous value the <code>gadget-default?</code> slot may have had.				

It is generally useful to set a default gadget in a frame, or a default menu if there is no suitable gadget.

When a default gadget is specified, using the default keyboard gesture in the frame invokes the activate callback for the default gadget. The default gesture is usually pressing the RETURN button.

See also [gadget-default?](#), page 493

## **gadget-documentation** *Generic function*

**Summary** Returns the documentation string for the specified gadget.

**Signature** `gadget-documentation gadget => documentation`

**Arguments** `gadget` An instance of type `<gadget>`.

**Values** `documentation` An instance of type `false-or(<string>)`.

**Description** Returns the documentation string for *gadget*.

The documentation string can be used to specify a short piece of online help text describing the action performed by the gadget. This text can then be displayed in a number of different ways. On Windows, for example, the documentation for a menu button might be displayed in the status bar of the application, and the documentation for a button might be displayed as a tooltip (a piece of pop-up text that appears next to the mouse pointer when the pointer is inside the region occupied by the gadget).

You are strongly encouraged to supply documentation strings for significant gadgets in your application. Because of the nature of their presentation, you should keep them as short as possible.

See also [gadget-documentation-setter](#), page 496

**gadget-documentation-setter** *Generic function*

**Summary** Sets the documentation string for the specified gadget.

**Signature** `gadget-documentation-setter documentation gadget => documentation`

**Arguments** `documentation` An instance of type `<string>`.

`gadget` An instance of type `<gadget>`.

**Values** `documentation` An instance of type `<string>`.

**Description** Sets the documentation string for *gadget* to *documentation*.

The documentation string can be used to specify a short piece of online help text describing the action performed by the gadget. This text can then be displayed in a number of different ways. On Windows, for example, the documentation for a menu button might be displayed in the status bar of the application, and the documentation for a button might be displayed as a tooltip (a piece of pop-up text that appears next to the mouse pointer when the pointer is inside the region occupied by the gadget).

You are strongly encouraged to supply documentation strings for significant gadgets in your application. Because of the nature of their presentation, you should keep them as short as possible.

**See also** `gadget-documentation`, page 495

`<status-bar>`, page 573

**gadget-enabled?** *Generic function*

**Summary** Returns true if the gadget is enabled.

**Signature** `gadget-enabled? gadget => enabled?`

Arguments	<i>gadget</i>	An instance of type < <i>gadget</i> >.
Values	<i>enabled?</i>	An instance of type < <i>boolean</i> >.
Description	Returns true if <i>gadget</i> is enabled. If the gadget is enabled, the user can interact with it in an appropriate way. If the gadget is not enabled, then the user cannot interact with it. For example, if a push button is not enabled, it cannot be clicked, or if a check box is not enabled, its setting cannot be switched on or off. Gadgets that are not enabled are generally grayed out on the screen.	
Example	<pre>*gadget* := contain(make     (&lt;radio-box&gt;,      items: range(from: 0, to: 20)));  gadget-enabled?(*gadget*);</pre>	
See also	<a href="#">&lt;<i>gadget</i>&gt;, page 485</a> <a href="#">gadget-enabled?-setter, page 497</a>	

**gadget-enabled?-setter***Generic function*

Summary	Toggles the enabled state of the specified gadget.	
Signature	<code>gadget-enabled?-setter enabled? gadget =&gt; enabled?</code>	
Arguments	<i>enabled?</i>	An instance of type < <i>boolean</i> >.
	<i>gadget</i>	An instance of type < <i>gadget</i> >.
Values	<i>enabled?</i>	An instance of type < <i>boolean</i> >.
Description	Causes <i>gadget</i> to become active (that is, available for input) or inactive, by toggling its enabled state. If <i>enabled?</i> is true, then <i>gadget</i> is enabled, otherwise, <i>gadget</i> is not enabled.	

If the gadget is enabled, the user can interact with it in an appropriate way. If the gadget is not enabled, then the user cannot interact with it. For example, if a push button is not enabled, it cannot be clicked, or if a check box is not enabled, its setting cannot be switched on or off. Gadgets that are not enabled are generally grayed out on the screen.

Example	<pre>*gadget* := contain(make                       (&lt;radio-box&gt;,                        items: range(from: 0, to: 20)));  gadget-enabled?(*gadget*) := #f;</pre>
See also	<p><a href="#">&lt;gadget&gt;</a>, page 485</p> <p><a href="#">gadget-enabled?</a>, page 496</p>

<b>gadget-id</b>	<i>Generic function</i>
Summary	Returns the ID of the specified gadget.
Signature	<b>gadget-id</b> <i>gadget</i> => <i>id</i>
Arguments	<b>gadget</b> An instance of type <a href="#">&lt;gadget&gt;</a> .
Values	<b><i>id</i></b> An instance of type <a href="#">&lt;object&gt;</a> .
Description	Returns the identifier of <i>gadget</i> . The identifier is typically a simple Dylan object that uniquely identifies the gadget. For most gadgets, it is usually not necessary. Making use of a gadget ID provides you with an additional way of controlling execution of your code, allowing you to create simple branching statements such as:

```

select (gadget-id)
  #"modify" => do-modify();
  #"add" => do-add();
  #"remove" => do-remove();
  #"done" => do-done();
end select;

```

In the specific case of tab controls, it is more important that you specify an ID. The gadget ID for a tab control is returned as the gadget value for that tab control.

**Example**

```

*gadget* := contain(make(<button>, id: #test,
                         label: "Test"));

gadget-id(*gadget*);

```

**See also**

[gadget-id-setter](#), page 499  
[gadget-value](#), page 517  
[<tab-control>](#), page 577

**gadget-id-setter***Generic function*

**Summary** Sets the ID of the specified gadget.

**Signature** `gadget-id-setter id gadget => id`

**Arguments** `id` An instance of type `<object>`.  
`gadget` An instance of type `<gadget>`.

**Values** `id` An instance of type `<object>`.

**Description** Sets the identifier of *gadget*. The identifier is typically a simple Dylan object that uniquely identifies the gadget. For most gadgets, it is usually not necessary, though it does provide you with an additional way of controlling execution of your code based on the gadget returned.

In the specific case of tab controls, it is more important that you specify an ID. The gadget ID for a tab control is returned as the gadget value for that tab control.

**Example**

```
*gadget* := contain(make(<button>, id: #test,
                           label: "Test"));

gadget-id(*gadget*) := #test-two;

gadget-id(*gadget*);
```

**See also**

[gadget-id](#), page 498  
[gadget-value](#), page 517  
[<tab-control>](#), page 577

	<i>Generic function</i>
<b>gadget-items</b>	
Summary	Returns the items for the specified gadget.
Signature	<code>gadget-items <i>gadget</i> =&gt; <i>items</i></code>
Arguments	<code><i>gadget</i></code> An instance of type <code>&lt;collection-gadget&gt;</code> .
Values	<code><i>items</i></code> An instance of type <code>&lt;sequence&gt;</code> . Default value: <code>#[]</code> .
Description	Returns the items for <i>gadget</i> . The items of any collection gadget is the collection of items that the collection gadget contains. In a list box, for example, the items are the list items themselves.
Example	The following code creates a list box whose items are the lower-cased equivalents of the symbols stated. Note that the label key for a gadget is a function that computes the label for the items in that gadget.

```
*gadget* := contain
  (make(<list-box>,
        items: #(#"One", #'Two", #'Three"),
        label-key:
          method (symbol)
            as-lowercase
              (as(<string>, symbol)) end));
```

You can return the items in the gadget as follows:

```
gadget-items(*g*);
```

This returns the symbol: #(#"one", #'two", #'three").

See also [gadget-items-setter](#), page 501  
[gadget-label-key](#), page 504  
[gadget-selection](#), page 511  
[gadget-value-key](#), page 521

## **gadget-items-setter**

*Generic function*

Summary	Sets the items for the specified gadget.				
Signature	<code>gadget-items-setter items gadget =&gt; items</code>				
Arguments	<table> <tr> <td><i>items</i></td><td>An instance of type &lt;sequence&gt;.</td></tr> <tr> <td><i>gadget</i></td><td>An instance of type &lt;collection-gadget&gt;.</td></tr> </table>	<i>items</i>	An instance of type <sequence>.	<i>gadget</i>	An instance of type <collection-gadget>.
<i>items</i>	An instance of type <sequence>.				
<i>gadget</i>	An instance of type <collection-gadget>.				
Values	<i>items</i> An instance of type <sequence>.				
Description	Sets the items for <i>gadget</i> to the items specified by <i>items</i> .				
Example	<pre>*gadget* := contain(make   (&lt;radio-box&gt;,    items: range(from: 0, to: 20)));  gadget-items(*gadget*) := range(from: 0, to: 15);</pre>				

See also [gadget-items](#), page 500

## gadget-key-press-callback

## Generic function

Summary	Returns the key-press callback for the specified gadget.
Signature	<code>gadget-key-press-callback gadget =&gt; key-press-callback</code>
Arguments	<code>gadget</code> An instance of type <code>&lt;collection-gadget&gt;</code> .
Values	<code>key-press-callback</code> An instance of type <code>false-or(&lt;command&gt;, &lt;function&gt;)</code> .
Description	Returns the key-press callback for <i>gadget</i> . The key-press callback is the callback invoked when a key on the keyboard is pressed while the gadget has focus. They are of most use in tab controls, list controls, table controls, graph controls, and tree controls.  In Windows, a good use for the key-press callback would be to mirror the behavior of Windows Explorer, where typing a filename, or part of a filename, selects the first file in the current folder whose name matches that typed.
See also	<code>gadget-key-press-callback-setter</code> , page 503 <code>&lt;list-control&gt;</code> , page 531 <code>&lt;tab-control&gt;</code> , page 577 <code>&lt;table-control&gt;</code> , page 586 <code>&lt;tree-control&gt;</code> , page 598

## **gadget-key-press-callback-setter** *Generic function*

Summary	Sets the key-press callback for the specified gadget.
Signature	<code>gadget-key-press-callback-setter <b>key-press-callback</b> <b>gadget</b> =&gt; <b>key-press-callback</b></code>
Arguments	<p><b>key-press-callback</b>  An instance of type <code>false-or(&lt;command&gt;, &lt;function&gt;)</code>.</p> <p><b>gadget</b> An instance of type <code>&lt;collection-gadget&gt;</code>.</p>
Values	<p><b>key-press-callback</b>  An instance of type <code>false-or(&lt;command&gt;, &lt;function&gt;)</code>.</p>
Description	<p>Sets the key-press callback for <i>gadget</i>. The key-press callback is the callback invoked when a key on the keyboard is pressed while the gadget has focus. They are of most use in tab controls, list controls, table controls, graph controls, and tree controls.</p> <p>In Windows, a good use for the key-press callback would be to mirror the behavior of Windows Explorer, where typing a filename, or part of a filename, selects the first file in the current folder whose name matches that typed.</p>
See also	<p><code>gadget-key-press-callback</code>, page 502</p> <p><code>&lt;list-control&gt;</code>, page 531</p> <p><code>&lt;tab-control&gt;</code>, page 577</p> <p><code>&lt;table-control&gt;</code>, page 586</p> <p><code>&lt;tree-control&gt;</code>, page 598</p>

**gadget-label** *Generic function*

Summary	Returns the label for the specified gadget.	
Signature	<code>gadget-label <i>gadget</i> =&gt; <i>label</i></code>	
Arguments	<i>gadget</i>	An instance of type <code>&lt;gadget&gt;</code> .
Values	<i>label</i>	An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code> .
Description	Returns the label for <i>gadget</i> .	
Example	<pre>*gadget* := contain(make(&lt;button&gt;, label: "Hello"));  gadget-label(*gadget*);</pre>	
See also	<a href="#">gadget-label-key</a> , page 504 <a href="#">gadget-label-setter</a> , page 505	

**gadget-label-key** *Generic function*

Summary	Returns the function that is used to compute the label for the items in the specified gadget.	
Signature	<code>gadget-label-key <i>gadget</i> =&gt; <i>label-key</i></code>	
Arguments	<i>gadget</i>	An instance of type <code>&lt;collection-gadget&gt;</code> .
Values	<i>label-key</i>	An instance of type <code>&lt;function&gt;</code> .
Description	Returns the function that is used to compute the labels for the items in <i>gadget</i> . Using a label key can be a useful way of consistently specifying labels that are a mapping of, but not directly equivalent to, the item names. As shown in the	

example, it is possible to force the case of item labels, and this is useful if the items are specified as symbol names, rather than strings.

**Example**

The following code creates a list box whose items are the lower-cased equivalents of the symbols stated.

```
*gadget* := contain
  (make(<list-box>,
        items: #(#"One", #"Two", #'Three"),
        label-key:
          method (symbol)
            as-lowercase
              (as(<string>, symbol))
            end));

```

The label key function can be returned as follows:

```
gadget-label-key(*gadget*);
```

**See also**

`gadget-label`, page 504

`gadget-label-setter`, page 505

`gadget-value-key`, page 521

**gadget-label-setter***Generic function***Summary**

Sets the label for the specified gadget.

**Signature**

`gadget-label-setter label gadget => label`

**Arguments**

*label* An instance of type `type-union(<string>, <image>)`.

*gadget* An instance of type `<gadget>`.

**Values**

*label* An instance of type `type-union(<string>, <image>)`.

Description	Sets the label for <i>gadget</i> to <i>label</i> . The <i>label</i> must be #f, a string, or an instance of <image>. Changing the label of a gadget may result in invoking the layout protocol on the gadget and its ancestor sheets, if the new label occupies a different amount of space than the old label.
Example	<pre>*gadget* := contain(make(&lt;button&gt;, label: "Hello"));  gadget-label(*gadget*) := "Hello world";</pre>
See also	<a href="#">gadget-label</a> , page 504 <a href="#">gadget-label-key</a> , page 504

## **gadget-mnemonic** *Generic function*

Summary	Returns the mnemonic for the specified gadget.	
Signature	<code>gadget-mnemonic <i>gadget</i> =&gt; <i>mnemonic</i></code>	
Arguments	<i>gadget</i>	An instance of type <gadget>.
Values	<i>mnemonic</i>	An instance of type <code>false-</code> <code>or(&lt;character&gt;)</code> .
Description	Returns the mnemonic for <i>gadget</i> . On Windows, the mnemonic is displayed as an underlined character in the label of the gadget, and pressing the key for that character activates the gadget or gives it the focus.	
See also	<a href="#">gadget-accelerator</a> , page 488 <a href="#">gadget-mnemonic-setter</a> , page 506	

## **gadget-mnemonic-setter** *Generic function*

Summary	Sets the mnemonic for the specified gadget.
---------	---

Signature	<code>gadget-mnemonic-setter mnemonic gadget =&gt; mnemonic</code>	
Arguments	<i>mnemonic</i>	An instance of type <code>false-or(&lt;character&gt;)</code> .
	<i>gadget</i>	An instance of type <code>&lt;gadget&gt;</code> .
Values	<i>mnemonic</i>	An instance of type <code>false-or(&lt;character&gt;)</code> .
Description	Sets the mnemonic for <i>gadget</i> to <i>mnemonic</i> . On Windows, the mnemonic is displayed as an underlined character in the label of the gadget, and pressing the key for that character activates the gadget or gives it the focus.	
See also	<a href="#">gadget-accelerator-setter</a> , page 489 <a href="#">gadget-mnemonic</a> , page 506	

## gadget-orientation

*Generic function*

Summary	Returns the orientation of the specified gadget.
Signature	<code>gadget-orientation gadget =&gt; orientation</code>
Arguments	<i>gadget</i>
Values	<i>orientation</i>
	An instance of type <code>type-one-of(#"horizontal", #"vertical", #"none")</code> .
Description	Returns the orientation of <i>gadget</i> : either horizontal or vertical.
Example	The following code creates a vertical row of buttons:

```
*buttons* := contain(make(<button-box>,
    selection-mode: #"multiple",
    orientation: #"vertical",
    items: range(from: 0, to: 5)));
```

The orientation can be returned as follows:

```
gadget-orientation(*buttons*);
```

## **gadget-popup-menu-callback**

*Generic function*

Summary	Returns the popup menu callback of the specified gadget.
Signature	<b>gadget-popup-menu-callback</b> <i>gadget</i> => <i>popup-menu-callback</i>
Arguments	<i>gadget</i> An instance of type < <b>gadget</b> >.
Values	<i>popup-menu-callback</i> An instance of type < <b>function</b> >.
Description	Returns the popup menu callback of <i>gadget</i> . This is typically a function that is used to create a context-sensitive menu of available commands. It is generally invoked when the user right clicks on the gadget.
See also	<b>gadget-popup-menu-callback-setter</b> , page 508

## **gadget-popup-menu-callback-setter**

*Generic function*

Summary	Sets the popup menu callback of the specified gadget.
Signature	<b>gadget-popup-menu-callback-setter</b> <i>popup-menu-callback</i> <i>gadget</i> => <i>popup-menu-callback</i>
Arguments	<i>popup-menu-callback</i> An instance of type < <b>function</b> >.

	<i>gadget</i>	An instance of type < <b>gadget</b> >.
Values	<i>popup-menu-callback</i>	An instance of type < <b>function</b> >.
Description		Sets the popup menu callback of <i>gadget</i> to <i>function</i> . The function should typically create a menu of commands suited to the context in which the function is called. The function is generally invoked by right-clicking on the gadget.
See also		<code>gadget-popup-menu-callback</code> , page 508

## **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.

## **gadget-read-only?** *Generic function*

Summary	Returns true if the gadget is editable.
Signature	<code>gadget-read-only? gadget =&gt; read-only?</code>
Arguments	<code>gadget</code> An instance of type <code>&lt;gadget&gt;</code> .
Values	<code>read-only?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>gadget</i> is read-only. The read-only attribute of a gadget is of most use with text gadgets.
See also	<a href="#">gadget-enabled?, page 496</a>

## **gadget-scrolling-horizontally?** *Generic function*

Summary	Returns true if the specified gadget has an associated horizontal scroll bar.
Signature	<code>gadget-scrolling-horizontally? gadget =&gt; horizontal?</code>
Arguments	<code>gadget</code> An instance of type <code>&lt;gadget&gt;</code> .
Values	<code>horizontal?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if the <i>gadget</i> has an associated horizontal scroll bar, false otherwise.
See also	<a href="#">gadget-scrolling-vertically?, page 510</a>

## **gadget-scrolling-vertically?** *Generic function*

Summary	Returns true if the specified gadget has an associated vertical scroll bar.
---------	---

Signature	<code>gadget-scrolling-vertically? gadget =&gt; vertical?</code>	
Arguments	<code>gadget</code>	An instance of type <code>&lt;gadget&gt;</code> .
Values	<code>vertical?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if the <i>gadget</i> has an associated vertical scroll bar, false otherwise.	
See also	<a href="#">gadget-scrolling-horizontally?, page 510</a>	

**gadget-selection***Generic function*

Summary	Returns the currently selected items of the specified gadget.	
Signature	<code>gadget-selection gadget =&gt; selection</code>	
Arguments	<code>gadget</code>	An instance of type <code>&lt;collection-gadget&gt;</code> .
Values	<code>selection</code>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;integer&gt;)</code> . Default value: <code>#[]</code> .
Description	Returns the keys for the currently selected items of <i>gadget</i> . Generally, you should use <code>gadget-value</code> to return the selected item, rather than <code>gadget-selection</code> , which is best used for handling repeated items.  Single selection gadgets (such as radio boxes) always have exactly one key selected. Multiple selection gadgets (such as check boxes) have zero or more keys selected. The value of a collection gadget is determined by calling the <code>value</code> key of the gadget on each selected item in the gadget.	
Example	Create a radio box as follows:	
	<pre>*radio* := contain(make(&lt;radio-box&gt;,                       items: range(from: 0, to: 5)));</pre>	

Select one of the items in the radio box. This selection can be returned with:

```
gadget-selection(*radio*);
```

See also

**gadget-items**, page 500

**gadget-selection-mode**, page 512

**gadget-selection-setter**, page 513

**gadget-value**, page 517

## **gadget-selection-mode**

*Generic function*

Summary	Returns the type of selection for the specified gadget.
Signature	<b>gadget-selection-mode</b> <i>gadget</i> => <b>selection-mode</b>
Arguments	<i>gadget</i> An instance of type <collection-gadget>.
Values	<i>selection-mode</i> An instance of <b>type one-of(#"single", "#multiple", "#none")</b> .
Description	Returns the selection mode for <i>gadget</i> . Typically, gadgets are either single or multiple selection (that is, either only one item can be selected at a time, or any number of items can be selected), or there is no selection behavior (items cannot be selected). Some gadgets, such as list boxes and button boxes, can choose a selection mode at initialization time using the <b>selection-mode:</b> init-keyword.
Example	Create a radio box as follows:

```
*radio* := contain(make(<radio-box>,
                      items: range(from: 0, to: 5)));
```

The selection mode of the radio box is returned with:

```
gadget-selection-mode(*radio*);
```

Because the gadget is a radio box, only one item of which may be selected at a time, the selection mode returned is `#"single"`.

See also [<button-box>](#), page 473

[gadget-selection](#), page 511

[gadget-selection-setter](#), page 513

[<list-box>](#), page 529

## **gadget-selection-setter** *Generic function*

**Summary** Sets the selection of the specified gadget.

**Signature** `gadget-selection-setter selection gadget #key do-callback? => selection`

**Arguments** `selection` An instance of type `limited(<sequence>, of: <integer>)`.

`gadget` An instance of type `<collection-gadget>`.

`do-callback?` An instance of type `<boolean>`. Default value: `#f`.

**Values** `selection` An instance of type `limited(<sequence>, of: <integer>)`.

**Description** Sets the selection of *gadget*. When setting the selection, you need to be wary of the selection mode for *gadget*. It is an error to try to set multiple items in a single selection mode gadget.

If *do-callback?* is true, the selection callback for *gadget* is invoked.

As with `gadget-selection`, you should usually use `gadget-value-setter` to set the selected item, rather than `gadget-selection-setter`, which is best used for handling repeated items. See `gadget-selection`, page 511 for more details.

**Example** Create a radio box as follows:

```
*radio* := contain(make(<radio-box>,
                        items: range(from: 0, to: 5)));
```

You can select the third item with:

```
gadget-selection(*radio*, do-callback?: #t) := #[3];
```

This sets the appropriate item, and invokes the callback that would have been invoked had the item been set manually, rather than programmatically (assuming that such a callback has been defined).

**See also** `gadget-selection`, page 511

`gadget-selection-mode`, page 512

`gadget-value-setter`, page 524

## gadget-slug-size

*Generic function*

**Summary** Returns the slug size of the specified gadget.

**Signature** `gadget-slug-size gadget => slug-size`

**Arguments** `gadget` An instance of type `<scroll-bar>`.

**Values** `slug-size` An instance of type `<real>`.

**Description** Returns the slug size of `gadget`. The slug is the part of `gadget` that can be dragged using the mouse. The value returned uses the same units as those used for `gadget-value-range`.

**Note:** The Microsoft Windows Interface Guidelines refer to the slug as a *scroll-box*, and the area in which the slug can slide as the *scroll-shaft*. You should be aware of this difference if you are using those guidelines as a reference.

See also [gadget-slug-size-setter](#), page 515  
[gadget-value-range](#), page 522

## gadget-slug-size-setter *Generic function*

Summary	Sets the slug size of the specified gadget.
Signature	<code>gadget-slug-size-setter slug-size gadget =&gt; slug-size</code>
Arguments	<code>slug-size</code> An instance of type <code>&lt;real&gt;</code> . <code>gadget</code> An instance of type <code>&lt;gadget&gt;</code> .
Values	<code>slug-size</code> An instance of type <code>&lt;real&gt;</code> .
Description	Sets the slug size of <i>gadget</i> . The value should use the same units as those used for <a href="#">gadget-value-range</a> .  <b>Note:</b> The Microsoft Windows Interface Guidelines refer to the slug as a <i>scroll-box</i> , and the area in which the slug can slide as the <i>scroll-shaft</i> . You should be aware of this difference if you are using those guidelines as a reference.

See also [gadget-slug-size](#), page 514

## gadget-test *Generic function*

Summary	Returns the test function for the specified gadget.
Signature	<code>gadget-test gadget =&gt; gadget-test</code>

Arguments	<i>gadget</i>	An instance of type <collection-gadget>.
Values	<i>gadget-test</i>	An instance of type <function>.
Description		Returns the test function for the specified gadget. This function is used to test whether two items of the collection are considered identical.

## **gadget-text** *Generic function*

Summary	Returns the text for the specified gadget.	
Signature	<b>gadget-text</b> <i>gadget</i> => <i>gadget-text</i>	
Arguments	<i>gadget</i>	An instance of type <text-gadget>.
Values	<i>gadget-text</i>	An instance of type <string>.
Description	Returns the text for the specified gadget.	
Example	First, create and display a text field by typing the following into an interactor:	

```
*g* := contain(make(<text-field>,
                     value-type: <integer>));
```

Next, type something into the text field. You can return the text string that you just typed with the following form:

```
gadget-text(*g*);
```

See also	<b>gadget-text-setter</b> , page 517 <b>&lt;text-gadget&gt;</b> , page 595
----------	---

## **gadget-text-setter** *Generic function*

Summary	Sets the text for the specified gadget.	
Signature	<code>gadget-text gadget-text gadget =&gt; gadget-text</code>	
Arguments	<i>gadget-text</i>	An instance of type <code>&lt;string&gt;</code> .
	<i>gadget</i>	An instance of type <code>&lt;text-gadget&gt;</code> .
Values	<i>gadget-text</i>	An instance of type <code>&lt;string&gt;</code> .
Description	Sets the text for the specified gadget.	
Example	First, create and display a text field by typing the following into an interactor:	
	<pre>*g* := contain(make(&lt;text-field&gt;,                      value-type: &lt;integer&gt;));</pre>	
	Next, set the value of the text field with the following form:	
	<pre>gadget-text-setter("Hello world", *g*);</pre>	
See also	<a href="#">gadget-text</a> , page 516 <a href="#">&lt;text-gadget&gt;</a> , page 595	

## **gadget-value** *Generic function*

Summary	Returns the gadget value of the specified gadget.	
Signature	<code>gadget-value gadget =&gt; gadget-value</code>	
Arguments	<i>gadget</i>	An instance of type <code>&lt;value-gadget&gt;</code> .
Values	<i>gadget-value</i>	An instance of type <code>&lt;object&gt;</code> .
Description	Returns the gadget value of the specified gadget.	

The interpretation of the value varies from gadget to gadget. Most gadgets conceptually have “raw” values that can be determined directly using the generic function appropriate to the gadget class concerned (`gadget-text` for an instance of `<text-gadget>`, `gadget-selection` for an instance of `<collection-gadget>`, and so on). These gadget classes also have a convenience method on `gadget-value` that wraps up the raw value in some useful way. So, text gadgets have a method on `gadget-value` that converts the `gadget-text` based on the `gadget-value-type`, for example converting the string to an integer for `value-type: <integer>`.

The `gadget-value` method for collection gadgets is different for single and multiple selection gadgets. For single selection, the item that is selected is returned. For multiple selection, a sequence of the selected items is returned.

**Note:** If the gadget ID has been specified for a tab control, then this is returned as the gadget value.

#### Example

Create a radio button:

```
*radio* := contain(make(<radio-button>,
    label: "Radio"));
```

The gadget value of `*radio*` can be returned as follows:

```
gadget-value(*radio*);
```

If the radio button is selected, `gadget-value` returns `#t`. If not selected, `gadget-value` returns `#f`.

#### See also

`<gadget>`, page 485

`gadget-id`, page 498

`gadget-value-key`, page 521

`gadget-value-range`, page 522

`gadget-value-setter`, page 524

`gadget-value-type`, page 525

**gadget-value-changed-callback** *Generic function*

Summary	Returns the value-changed callback of the specified gadget.
Signature	<b>gadget-value-changed-callback</b> <i>gadget</i> => <b>value-changed-callback</b>
Arguments	<i>gadget</i> An instance of type <value-gadget>.
Values	<i>value-changed-callback</i> An instance of type <b>false-or(&lt;function&gt;)</b> .
Description	<p>Returns the value-changed callback of <i>gadget</i>. This is the call-back function that is called once the gadget value of <i>gadget</i> has been changed.</p> <p>The value-changed callback function is invoked with one argument, the gadget.</p> <p>If <b>gadget-value-changed-callback</b> returns #f, there is no value changed callback for <i>gadget</i>.</p>
See also	<b>gadget-value-changed-callback-setter</b> , page 519

**gadget-value-changed-callback-setter** *Generic function*

Summary	Sets the value-changed-callback of the specified gadget.
Signature	<b>gadget-value-changed-callback-setter</b> <i>callback gadget</i> => <b>callback</b>
Arguments	<i>callback</i> An instance of type <b>false-or(&lt;function&gt;)</b> . <i>gadget</i> An instance of type <gadget>.
Values	<i>callback</i> An instance of type <b>false-or(&lt;function&gt;)</b> .

Description	Sets the value-changed callback of <i>gadget</i> to <i>function</i> . This is the callback function that is called once the gadget value of <i>gadget</i> has been changed.  The value-changed callback function is invoked with one argument, the gadget.
See also	<a href="#">gadget-value-changed-callback</a> , page 519

## **gadget-value-changing-callback** *Generic function*

Summary	Returns the value changing callback of the specified gadget.
Signature	<code>gadget-value-changing-callback <i>gadget</i> =&gt; <i>value-changing-callback</i></code>
Arguments	<code><i>gadget</i></code> An instance of type <code>&lt;gadget&gt;</code> .
Values	<code><i>value-changing-callback</i></code> An instance of type <code>&lt;function&gt;</code> .
Description	Returns the function that will be called when the value of <i>gadget</i> is in the process of changing, such as when a slider is being dragged. The <i>function</i> will be invoked with a two arguments, <i>gadget</i> and the new value.
See also	<a href="#">gadget-value-changing-callback-setter</a> , page 520

## **gadget-value-changing-callback-setter** *Generic function*

Summary	Sets the value-changing callback of the specified gadget.
Signature	<code>gadget-value-changing-callback-setter <i>value-changing-callback</i> <i>gadget</i> =&gt; <i>value-chaning-callback</i></code>

Arguments	<i>value-changing-callback</i>
	An instance of type <function>.
<i>gadget</i>	An instance of type <gadget>.
Values	<i>value-changing-callback</i>
	An instance of type <function>.
Description	Sets the function that will be called when the value of <i>gadget</i> is in the process of changing, such as when a slider is being dragged. The <i>function</i> will be invoked with a two arguments, <i>gadget</i> and the new value.
See also	<a href="#">gadget-value-changing-callback</a> , page 520

## **gadget-value-key** *Generic function*

Summary	Returns the function that is used to calculate the gadget value of the specified gadget.
Signature	<code>gadget-value-key <i>gadget</i> =&gt; <i>value-key</i></code>
Arguments	<i>gadget</i> An instance of type <collection-gadget>.
Values	<i>value-key</i> An instance of type <function>. Default value: <code>identity</code> .
Description	Returns the function that is used to calculate the gadget value of <i>gadget</i> , given the selected items. The function takes an item and returns a value.
Example	The list box defined below has three items, each of which is a pair of two symbols. A label-key and a value-key is defined such that the label for each item is calculated from the first symbol in each pair, and the gadget value is calculated from the second.

```
*list* := contain(make(<list-box>,
                      items: #(#("One", "#one"),
                               ("Two", "#two"),
                               ("Three", "#three")),
                      label-key: first,
                      value-key: second));
```

This ensures that while the label of the first item is displayed on-screen as `one`, the value returned from that item is `#"one"`, and similarly for the other items in the list.

The gadget value key function can be returned with:

```
gadget-value-key(*list*);
```

See also      [gadget-label-key](#), page 504  
[gadget-value](#), page 517

## gadget-value-range

*Generic function*

Summary	Returns the range of values for the specified gadget.	
Signature	<code>gadget-value-range gadget =&gt; range</code>	
Arguments	<code>gadget</code>	An instance of type <code>&lt;value-range-gadget&gt;</code> .
Values	<code>range</code>	An instance of type <code>&lt;range&gt;</code> .
Description	Returns the range of values for <i>gadget</i> . The value range is the elements represented by the range specified for <i>gadget</i> .	
<p><b>Note:</b> The value range is not simply the difference between the maximum and minimum values in the range. Consider the following range:</p>		

```
range (from: 10, to: 0, by: -2)
```

In this case, the value range is the elements 10, 8, 6, 4, 2, 0.

The units in which the range is specified are also used for `gadget-slug-size`.

**Example** You can create a slider with a given range as follows:

```
*slider* := contain(make(<slider>,
                        value-range: range(from: -20,
                                            to: 20,
                                            by: 5)));
```

You can return the range of this gadget by executing the following:

```
gadget-value-range(*slider*);
```

which in this case returns {range -20 through 20, by 5}.

**See also** [gadget-slug-size](#), page 514

[gadget-value](#), page 517

[gadget-value-range-setter](#), page 523

## gadget-value-range-setter

*Generic function*

**Summary** Sets the range of values for the specified gadget.

**Signature** `gadget-value-range-setter range gadget => range`

**Arguments** `range` An instance of type <range>.

`gadget` An instance of type <value-range-gadget>.

**Values** `range` An instance of type <range>.

**Description** Sets the range of values for *gadget*. The value range is the elements represented by the range specified for *gadget*.

**Example** Create a slider without specifying a range:

```
*slider* := contain(make(<slider>));
```

You can specify the range of this gadget by executing the following:

```
gadget-value-range(*slider*) :=
  (range (from: -20 to: 20, by: 5));
```

See also [gadget-value-range](#), page 522

## **gadget-value-setter** *Generic function*

**Summary** Sets the gadget value of the specified gadget.

**Signature** **gadget-value-setter** *gadget-value* *gadget* #key *do-callback?* => *gadget-value*

**Arguments** *gadget-value* An instance of type <object>.

*gadget* An instance of type <value-gadget>.

*do-callback?* An instance of type <boolean>. Default value: #f.

**Values** *gadget-value* An instance of type <object>.

**Description** Sets the gadget value of *gadget*.

The *value* that you need to specify varies from gadget to gadget. For example, for a scroll bar, *value* might be a number between 0 and 1, while for a radio button, *value* is either true or false.

If *do-callback?* is true, the value-changed callback for *gadget* is invoked.

**Example** Create a radio button:

```
*radio* := contain(make(<radio-button>,
  label: "Radio"));
```

The gadget value of \*radio\* can be set with either of the following:

```
gadget-value(*radio*) := #t;
gadget-value(*radio*) := #f;
```

Setting the gadget value to `#t` selects the button, and setting it to `#f` deselects it.

See also [gadget-value](#), page 517

## **gadget-value-type** *Generic function*

Summary Returns the type of the gadget value for the specified gadget.

Signature `gadget-value-type gadget => type`

Arguments `gadget` An instance of type `<value-gadget>`.

Values `type` An instance of type `<type>`.

Description Returns the type of the gadget value for `gadget`.

Example The following code creates a text field, the contents of which are constrained to be an integer.

```
*numeric* := contain(make(<text-field>,
                           value-type: <integer>));
```

Evaluating the following code confirms the gadget value type to be the class `<integer>`.

```
gadget-value-type(*numeric*);
```

See also [gadget-value](#), page 517

## **gadget-x-alignment** *Generic function*

Summary Returns the horizontal alignment of the specified gadget.

Signature `gadget-x-alignment gadget => alignment`

Arguments `gadget` An instance of type `<gadget>`.

Values	<i>alignment</i>	An instance of type <code>one-of(#"left", #"right", #"center")</code> .
Description	Returns the horizontal alignment of <i>gadget</i> . You can only set the horizontal alignment of a gadget when first initializing that gadget, using the <code>x-alignment:</code> init-keyword.	
See also	<code>gadget-y-alignment</code> , page 526	

## **gadget-y-alignment** *Generic function*

Summary	Returns the vertical alignment of the specified gadget.	
Signature	<code>gadget-x-alignment gadget =&gt; alignment</code>	
Arguments	<i>gadget</i>	An instance of type <code>&lt;gadget&gt;</code> .
Values	<i>alignment</i>	An instance of type <code>one-of(#"top", #"bottom", #"center")</code> .
Description	Returns the vertical alignment of <i>gadget</i> . You can only set the vertical alignment of a gadget when first initializing that gadget, using the <code>y-alignment:</code> init-keyword.	
See also	<code>gadget-x-alignment</code> , page 525	

## **<group-box>** *Open abstract instantiable class*

Summary	The class of gadgets that group their children using a labelled border.	
Superclasses	<code>&lt;gadget&gt;</code>	
Init-keywords	<code>label:</code>	An instance of type <code>&lt;label&gt;</code> .

**label-position:**

An instance of type `one-of(#"top", "#"bottom")`. Default value: `#"top"`.

Description	The class of gadgets that group their children using a labelled border. You can use this gadget class to group together a number of related items visually.
-------------	---



**Figure 8.11** A group box

The `label:` init-keyword specifies a string or icon that is to be used as a label for the gadget.

The `label-position:` init-keyword is used to specify whether the label should be displayed along the top or the bottom edge of the border.

Internally, this class maps into the Windows group box control.

Operations	None.
------------	-------

Example	<pre>contain(make(&lt;group-box&gt;,             child: make(&lt;radio-box&gt;, items: #(1,2,3,4),                         orientation: #"vertical"),             label: "Select integer:"));</pre>
---------	---

See also	<a href="#">&lt;border&gt;</a> , page 469 <a href="#">&lt;check-box&gt;</a> , page 475 <a href="#">&lt;push-box&gt;</a> , page 552 <a href="#">&lt;radio-box&gt;</a> , page 556
----------	--

	<i>Generic function</i>
<b>item-object</b>	
Summary	Returns the Dylan object representing an item in a list or table control.
Signature	<code>item-object item =&gt; object</code>
Arguments	<code>item</code> An instance of type <code>type-union(&lt;list-item&gt;, &lt;table-item&gt;)</code> .
Values	<code>object</code> An instance of type <code>&lt;object&gt;</code> .
Description	Returns the Dylan object representing an item in a list or table control.
<b>&lt;label&gt;</b>	<i>Open abstract instantiable class</i>
Summary	The class of label gadgets.
Superclasses	<code>&lt;gadget&gt;</code>
Init-keywords	<code>label:</code> An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code> .
Description	<p>The class of label gadgets.</p> <p>The <code>label:</code> init-keyword specifies a string or image that is to be used as a label for the gadget. If you use an image, you should be wary of its size: only use images that are the size of a typical icon.</p> <p>Internally, this class maps into the Windows static control.</p>
Operations	<code>gadget-label gadget-label-setter make-menu-from-command-table-menu make-menus-from-command-table note-progress noting-progress</code>

Example      `contain(make(<label>, label: "Hello"));`

See also     [labelling](#), page 529

## **labelling** *Statement macro*

Summary     Creates the specified sheet and assigns a label to it.

Macro call    `labelling ([options]) {pane} end`

Arguments    *options*        Dylan arguments<sub>bnf</sub>

*pane*         A Dylan expression<sub>bnf</sub>

Values        None.

Description    Creates *pane* with a label assigned to it, taking into account any of the specified *options*.

The options specified may be any of the legal init-keywords used to specify an instance of `<label>`. If no options are specified, then the default label is used.

The *pane* is an expression whose return value is the sheet to which the label should be assigned.

Example      `labelling ("Color Type:")
 make(<check-box>, items: #("Color", "Monochrome"))
end;`

See also     [<label>](#), page 528

## **<list-box>** *Open abstract instantiable class*

Summary     The class of list boxes.

Superclasses    `<collection-gadget> <action-gadget>`

Init-keywords    **borders:**    An instance of type `one-of(#f, #"none", #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output")`. Default value: `#f`.

**scroll-bars:**    An instance of type `one-of(#f, #"none", #"horizontal", #"vertical", #"both", #"dynamic")`. Default value: `#"both"`.

Description    The class of list boxes.



The **borders:** init-keyword lets you specify a border around the list box. If specified, a border of the appropriate type is drawn around the gadget.

The **scroll-bars:** init-keyword lets you specify the presence of scroll bars around the gadget. By default, both horizontal and vertical scroll bars are created. You can also force the creation of only horizontal or vertical scroll bars, or you can create scroll bars dynamically: that is, have them created only if necessary, dependent on the size of the gadget. If **scroll-bars:** is `#f`, no scroll bars are added to the gadget.

Internally, this class maps into the Windows list box control.

Operations    None.

Example    The following creates a list of three items, without scroll bars.

```
*list* := contain(make(<list-box>,
                      items: #(("One", #"one"),
                               #("Two", #"two"),
                               #("Three", #"three")),
                      label-key: first,
                      value-key: second,
                      scroll-bars: #f));
```

See also      [`<list-control>`](#), page 531  
[`<list-item>`](#), page 537

## **<list-control>** *Open abstract instantiable class*

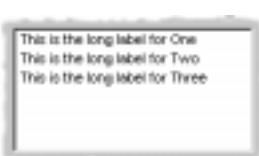
**Summary**      The class of list controls.

**Superclasses**    [`<collection-gadget>`](#) [`<action-gadget>`](#)

**Init-keywords**

- `icon-function:`** An instance of type [`<function>`](#).
- `view:`**       An instance of type [`<list-control-view>`](#).  
Default value: `#"list"`.
- `borders:`**    An instance of type `one-of(#f, #"none",  
#"flat", #"sunken", #"raised",  
#"ridge", #"groove", #"input",  
#"output")`. Default value: `#f`.
- `scroll-bars:`** An instance of type `one-of(#f, #"none",  
#"horizontal", #"vertical", #"both",  
#"dynamic")`. Default value: `"both"`.
- `popup-menu-callback:`**  
An instance of type [`<function>`](#).
- `key-press-callback:`**  
An instance of type `false-or(<command>,  
<function>)`.

**Description**      The class of list controls. These are controls that can list items in a number of different ways, using a richer format than the [`<list-box>`](#) class. Examples of list controls are the main panels in the Windows Explorer, or the Macintosh Finder. List controls can also be seen in the standard Windows 95 Open File dialog box.



The **icon-function:** init-keyword lets you specify a function to supply icons for display in the control. The function is called with the item that needs an icon as its argument, and it should return an instance of `<image>` as its result. Typically, you might want to define an icon function that returns a different icon for each kind of item in the control. For example, if the control is used to display the files and directories on a hard disk, you would want to return the appropriate icon for each registered file type.

The **view:** init-keyword can be used to specify the way in which the items in the list box are displayed. There are three options, corresponding to the view options that will be familiar to most users of GUI-based operating systems.

The **borders:** init-keyword lets you specify a border around the list control. If specified, a border of the appropriate type is drawn around the gadget.

The **scroll-bars:** init-keyword lets you specify the presence of scroll bars around the gadget. By default, both horizontal and vertical scroll bars are created. You can also force the creation of only horizontal or vertical scroll bars, or you can create scroll bars dynamically: that is, have them created only if necessary, dependent on the size of the gadget. If **scroll-bars:** is `#f`, no scroll bars are added to the gadget.

You can use the **popup-menu-callback:** init-keyword to specify a context-sensitive menu to display for one or more selected items in the list control. In Windows 95, for instance, such a context-sensitive menu can be displayed by right-clicking on any item or group of selected items in the list control.

The **key-press-callback:** init-keyword lets you specify a key-press callback. This type of callback is invoked whenever a key on the keyboard is pressed while the gadget has focus. See **gadget-key-press-callback**, page 502, for a fuller description of key-press callbacks.

Internally, this class maps into the Windows list view control.

Operations	<code>add-item</code> <code>find-item</code> <code>list-control-view</code> <code>list-control-view-setter</code> <code>make-item</code> <code>remove-item</code>
See also	<a href="#">add-item</a> , page 467 <a href="#">list-control-view</a> , page 535 <a href="#">make-item</a> , page 538 <a href="#">remove-item</a> , page 560

## list-control-icon-function *Generic function*

Summary	Returns the icon function for the specified list control.
Signature	<code>list-control-icon-function</code> <i>list-control</i> => <i>icon-function</i>
Arguments	<i>list-control</i> An instance of < <code>list-control</code> >.
Values	<i>icon-function</i> An instance of type < <code>function</code> >.
Description	<p>Returns the icon-function for <i>list-control</i>. This function lets you specify which icon to display for each item in the control. The function is called with the item that needs an icon as its argument, and it should return an instance of &lt;<code>image</code>&gt; as its result. Typically, you might want to define an icon function that returns a different icon for each kind of item in the control. For example, if the control is used to display the files and directories on a hard disk, you would want to return the appropriate icon for each registered file type.</p> <p>Note that, unlike tree controls, the icon function for a list control can be changed once the list control has been created.</p>

See also	<a href="#"><code>&lt;list-control&gt;</code></a> , page 531 <a href="#">list-control-icon-function-setter</a> , page 534
----------	--

		<i>Generic function</i>
Summary	Sets the icon function for the specified list control.	
Signature		<code>list-control-icon-function-setter <i>icon-function</i> <i>list-control</i> =&gt; <i>icon-function</i></code>
Arguments	<p><i>icon-function</i> An instance of type <code>&lt;function&gt;</code>.</p> <p><i>list-control</i> An instance of <code>&lt;list-control&gt;</code>.</p>	
Values	<i>icon-function</i> An instance of type <code>&lt;function&gt;</code> .	
Description	<p>Sets the icon-function for <i>list-control</i>. This function lets you specify which icon to display for each item in the control. The function is called with the item that needs an icon as its argument, and it should return an instance of <code>&lt;image&gt;</code> as its result. Typically, you might want to define an icon function that returns a different icon for each kind of item in the control. For example, if the control is used to display the files and directories on a hard disk, you would want to return the appropriate icon for each registered file type.</p> <p>Note that, unlike tree controls, the icon function for a list control can be changed once the list control has been created.</p>	
See also	<p><code>&lt;list-control&gt;</code>, page 531</p> <p><code>list-control-icon-function</code>, page 533</p>	

	<i>Type</i>
Summary	The type of possible views for a list control
Equivalent	<code>one-of(#"small-icon", #"large-icon", #"list")</code>
Superclasses	None.

Init-keywords	None.
Description	<p>This type represents the acceptable values for the view arguments to operators of <code>&lt;list-control&gt;</code>. You should not attempt to redefine this type in any way.</p> <p>There are three possible values, corresponding to the view options that will be familiar to most users of GUI-based operating systems:</p> <ul style="list-style-type: none"> <li><code>#"small-icon"</code> Displays each item in the list control using a small icon to the left of the item. Items are arranged horizontally.</li> <li><code>#"large-icon"</code> Displays each item in the list control using a large icon to the left of the item. Items are arranged horizontally.</li> <li><code>#"list"</code> Displays each item in the list control using a small icon to the left of the item. Items are arranged vertically in one column.</li> </ul>
See also	<p><code>&lt;list-control&gt;</code>, page 531</p> <p><code>list-control-view</code>, page 535</p> <p><code>&lt;table-control-view&gt;</code>, page 589</p>

		<i>Generic function</i>
Summary		Returns the view for the specified list control.
Signature		<code>list-control-view list-control =&gt; view</code>
Arguments	<code>list-control</code>	An instance of <code>&lt;list-control&gt;</code> .
Values	<code>view</code>	An instance of type <code>&lt;list-control-view&gt;</code> .

**Description** Returns the view for *list-control*. The view defines how items in the list control are displayed. Three views are available; items are accompanied either by a small icon or a large icon. In addition, items can be listed vertically, and additional details can be displayed for each item. For more details, see the description for `<list-control-view>`, page 534.

**Example** Given a list control created with the following code:

```
*list* := contain(make(<list-control>,
                      items: #(#("One", "#one"),
                               #("Two", "#two"),
                               #("Three", "#three")),
                      view: #"list"
                      scroll-bars: #f));
```

The list control view may be returned with:

```
list-control-view(*list*);
```

**See also** `<list-control>`, page 531

`<list-control-view>`, page 534

`list-control-view-setter`, page 536

## list-control-view-setter

*Generic function*

**Summary** Sets the view for the specified list control.

**Signature** `list-control-view-setter view list-control => view`

**Arguments** `view` An instance of type `<list-control-view>`.

`list-control` An instance of `<list-control>`.

**Values** `view` An instance of type `<list-control-view>`.

**Description** Sets the view for *list-control*. The view defines how items in the list control are displayed. Three views are available; items are accompanied either by a small icon or a large icon. In addition, items can be listed vertically, and additional details can be displayed for each item. For more details, see the description for `<list-control-view>`, page 534.

**Example** Given a list control created with the following code:

```
*list* := contain(make(<list-control>,
                      items: #("One",
                                "Two",
                                "Three")));
```

The list control view may be specified with:

```
list-control-view(*list*) := #"view";
```

**See also** `<list-control>`, page 531  
`<list-control-view>`, page 534  
`list-control-view`, page 535

## **<list-item>** *Open abstract instantiable class*

**Summary** The class that represents an item in a list control.

**Superclasses** `<object>`

**Init-keywords** `object:` An instance of type `<object>`. Default value: `#f.`

**Description** The class that represents an item in a list control.

**Operations** `add-item item-object remove-item`

**See also** `<list-control>`, page 531  
`<table-item>`, page 592

<b>make-item</b>	<i>Generic function</i>
Summary	Creates an item which can be inserted in the specified list control or table control.
Signature	<code>make-item list-or-table object #key frame-manager =&gt; item</code>
Arguments	<p><i>list-or-table</i>      An instance of <code>type-union (&lt;list-control&gt;, &lt;table-control&gt;).</code></p> <p><i>object</i>                An instance of type <code>&lt;object&gt;</code>.</p> <p><i>frame-manager</i>        An instance of type <code>&lt;frame-manager&gt;</code>.</p>
Values	<i>item</i> An instance of type <code>&lt;list-item&gt;</code> .
Description	<p>Creates an item that represents <i>object</i> which can be inserted in the specified <i>list-or-table</i>. To insert the item in the list control or table control, <code>add-item</code> is used. You would not normally call <code>make-item</code> explicitly: just use <code>add-item</code> and the item is created automatically before it is added to the list or table control.</p> <p>If the <i>frame-manager</i> argument is specified, then this is used instead of the default frame manager.</p>
See also	<p><code>add-item</code>, page 467</p> <p><code>find-item</code>, page 484</p> <p><code>&lt;list-control&gt;</code>, page 531</p> <p><code>&lt;list-item&gt;</code>, page 537</p> <p><code>remove-item</code>, page 560</p> <p><code>&lt;table-control&gt;</code>, page 586</p> <p><code>&lt;table-item&gt;</code>, page 592</p>

**make-menu-from-items***Generic function*

**Summary** Returns a menu object created from the specified items.

**Signature** ***make-menu-from-items*** *framem* *items* #key *owner* *title* *label-key*  
*value-key* *foreground* *background* *text-style* => *menu*

**Arguments** *framem* An instance of type <**frame-manager**>.

*items* An instance of type <**sequence**>.

*owner* An instance of type <**sheet**>.

*title* An instance of type <**string**>.

*label-key* An instance of <**function**>. Default value:  
**identity**.

*value-key* An instance of <**function**>. Default value:  
**identity**.

*foreground* An instance of type **false-or(<ink>)**.  
Default value: #f.

*background* An instance of type **false-or(<ink>)**.  
Default value: #f.

*text-style* An instance of type <**text-style**>.

**Values** *menu* An instance of type <**menu**>.

**Description** Returns a menu object created from the specified *items*.

The *framem* argument lets you specify a frame manager.

The *owner* argument is used to specify which sheet owns the menu. If you fail to supply this, then the menu will be owned by the entire screen.

You can specify a *title*, if desired.

The *label-key* and *value-key* can be functions used to compute the label and value for each item in the menu, respectively. For more information, see [gadget-label-key](#), page 504, or [gadget-value-key](#), page 521. In general, the label key can be trusted to “do the right thing” by default.

The *text-style* argument specified a text style for the menu. The *foreground* and *background* arguments specify foreground and background colors for the menu: *foreground* being used for the text in the menu, and *background* for the menu itself.

See also      [display-menu](#), page 483

## **make-node** *Generic function*

Summary	Creates a node which can be inserted in the specified tree control.	
Signature	<code>make-node tree object #key #all-keys =&gt; node</code>	
Arguments	<i>tree</i>	An instance of <code>&lt;tree-control&gt;</code> .
	<i>object</i>	An instance of type <code>&lt;object&gt;</code> .
Values	<i>node</i> An instance of type <code>&lt;tree-node&gt;</code> .	
Description	Creates a node that represents <i>object</i> which can be inserted in the specified <i>tree</i> . To insert the item in the tree control, <a href="#">add-node</a> is used. You would not normally call <code>make-node</code> explicitly: just use <code>add-node</code> and the node is created automatically before it is added to the tree control.	
See also	<a href="#">add-node</a> , page 468 <a href="#">find-node</a> , page 485 <a href="#">remove-node</a> , page 560 <a href="#">&lt;tree-control&gt;</a> , page 598	

## <menu>

*Open abstract instantiable class*

Summary	The class of menu gadgets.
Superclasses	<gadget> <multiple-child-composite-pane>
Init-keywords	<p><b>update-callback:</b></p> <p>An instance of type <code>false-or(&lt;function&gt;)</code>.</p> <p><b>owner:</b> An instance of type <code>&lt;sheet&gt;</code>.</p> <p><b>mnemonic:</b> An instance of type <code>false-or(&lt;character&gt;)</code>. Default value: <code>#f</code>.</p> <p><b>command:</b> An instance of <code>false-or(&lt;command&gt;)</code>. Default value: <code>#f</code>.</p>
Description	<p>The class of menu gadgets.</p> <p>Support for dynamically modifying the contents of a menu is provided in the form of an update callback. If this is supplied using the <code>update-callback:</code> init-keyword, then it is invoked just before the menu is displayed. This callback is free to make changes to the contents of the menu, which will then appear when the update callback is complete. Note that you can also supply an update callback to any menu box which forms a part of the menu, using the relevant init-keyword to <code>&lt;menu-box&gt;</code>.</p> <p>The <code>owner:</code> argument is used to specify which sheet owns the menu. If you fail to supply this, then the menu will be owned by the entire screen.</p> <p>The <code>mnemonic:</code> init-keyword is used to specify a keyboard mnemonic for the button. This is a key press that involves pressing the ALT key followed by a number of alphanumeric keys.</p> <p>The <code>command:</code> init-keyword specifies a command that is invoked when the menu is chosen. For most menus, you should not specify a command; instead, you assign menu</p>

buttons as children to the menu, and the menu buttons themselves have commands specified. However, in the rare case where the menu has no children, and you want the menu itself to invoke a command, you can use this init-keyword. Internally, this class maps into the menu Windows control.

Operations	<code>add-command choose-from-dialog choose-from-menu display-menu menu-owner</code>
Example	The following code creates a menu, <b>Hello</b> , that contains a single button, <b>World</b> . Notice how using <code>contain</code> creates a menu bar for you automatically. You should note that using <code>display-menu</code> would not have this effect.
	<pre>*menu* := contain(make(&lt;menu&gt;,                         label: "Hello",                         children:                           vector                             (make(&lt;menu-button&gt;,                                   label: "World")));</pre>

See also      `display-menu`, page 483  
               `make-menu-from-items`, page 539

<b>&lt;menu-bar&gt;</b>		<i>Open abstract instantiable class</i>
Summary		The class of menu bar gadgets.
Superclasses		<code>&lt;value-gadget&gt; &lt;multiple-child-composite-pane&gt;</code>
Init-keywords		<code>update-callback:</code> An instance of type <code>&lt;function&gt;</code> .
Description		The class of menu bar gadgets. Internally, this class maps into the Windows menu control.

Operations      **frame-menu-bar-setter make**

Example      The following example is similar to the example for `<menu>`, except that here, the menu bar object is explicitly defined. In the example for `<menu>`, it is created automatically by using `contain`:

```
*menu* := make(<menu-bar>,
               children:
                 vector(make(<menu>,
                             label: "Hello",
                             children: vector
                               (make(<menu-button>,
                                     label: "World")
                                ))));

```

See also      `<menu>`, page 541

## **<menu-box>** *Open abstract instantiable class*

Summary      A class that groups menu buttons.

Superclasses      `<collection-gadget>`

Init-keywords      **update-callback:**  
An instance of type `false-or(<function>)`.

Description      A class that groups menu buttons. Like the `<button-box>` class, you can use this class to create groups of menu buttons that are related in some way. A visual separator is displayed in the menu in which a menu box is inserted, separating the menu buttons defined in the menu box from other menu buttons or menu boxes in the menu.

An example of the way in which a menu box may be used is to implement the clipboard menu commands usually found in applications. A menu box containing items that represent the **Cut**, **Copy**, and **Paste** commands can be created and inserted into the **Edit** menu.

Internally, this class maps into the menu Windows control.

Support for dynamically modifying the contents of a menu box is provided in the form of an update callback. If this is supplied using the `update-callback:` init-keyword, then it is invoked just before the menu box is displayed (this usually occurs at the same time that the menu of which the menu box is a part is displayed). This callback is free to make changes to the contents of the menu box, which will then appear when the update callback is complete.

**Operations**      None.

**Example**

```
*menu-box* := contain(make(<menu-box>,
                           items: range
                           (from: 0, to: 5))),
```

**See also**      [<check-menu-box>](#), page 477

[<push-menu-box>](#), page 554

[<radio-menu-box>](#), page 557

## <menu-button>

*Open abstract instantiable class*

**Summary**      The class of all buttons that can appear in menus.

**Superclasses**      [<button>](#)

**Init-keywords**      `update-callback:`

An instance of type [<function>](#).

Description	The class of all buttons that can appear on menus.  You should take special care to define keyboard accelerators and keyboard mnemonics for any menu buttons you create. For a full discussion on this, see the entry for <button>, page 470  Internally, this class maps into the menu item Windows control.
Operations	None.
Example	<pre>contain   (make(&lt;menu-button&gt;, label: "Hello",         activate-callback:           method (gadget)             notify-user               (format-to-string                 ("Pressed button %=", gadget),                 owner: gadget) end));</pre>
See also	<a href="#">&lt;check-menu-button&gt;</a> , page 478 <a href="#">gadget-accelerator</a> , page 488 <a href="#">&lt;menu-box&gt;</a> , page 543 <a href="#">&lt;push-menu-button&gt;</a> , page 555 <a href="#">&lt;radio-menu-button&gt;</a> , page 558

	<i>Generic function</i>
Summary	Returns the sheet that owns the specified menu.
Signature	<code>menu-owner menu =&gt; sheet</code>
Arguments	<code>menu</code> An instance of type < <code>menu</code> >.
Values	<code>sheet</code> An instance of type < <code>sheet</code> >.

Description	Returns the sheet that owns <i>menu</i> , that is, the sheet in which <i>menu</i> is displayed.  Every menu should specify which sheet it is owned by. If this is not specified, then the menu will be owned by the entire screen.
-------------	--

**node-children** *Generic function*

Summary	Returns the children of the specified node in a tree control.
Signature	<code>node-children <i>tree-node</i> =&gt; <i>children</i></code>
Arguments	<code><i>tree-node</i></code> An instance of type <code>&lt;tree-node&gt;</code> .
Values	<code><i>children</i></code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;tree-node&gt;)</code> .
Description	Returns the children of <i>tree-node</i> in a tree control.
See also	<code>node-children-setter</code> , page 546  <code>node-parents</code> , page 548  <code>tree-control-children-generator</code> , page 602  <code>&lt;tree-node&gt;</code> , page 607

**node-children-setter** *Generic function*

Summary	Sets the children of the specified node in a tree control.
Signature	<code>node-children-setter <i>children tree-node</i> =&gt; <i>children</i></code>
Arguments	<code><i>children</i></code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;tree-node&gt;)</code> .  <code><i>tree-node</i></code> An instance of type <code>&lt;tree-node&gt;</code> .

Values	<i>children</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;tree-node&gt;)</code> .
Description	Sets the children of <i>tree-node</i> in a tree control.	
See also	<a href="#">node-children</a> , page 546 <a href="#">node-parents</a> , page 548 <a href="#">tree-control-children-generator</a> , page 602 <a href="#">&lt;tree-node&gt;</a> , page 607	

## **node-expanded?** *Generic function*

Summary	Returns true if the specified node is expanded in a tree control.	
Signature	<code>node-expanded? tree-node =&gt; expanded?</code>	
Arguments	<i>tree-node</i>	An instance of type <code>&lt;tree-node&gt;</code> .
Values	<i>expanded?</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>tree-node</i> is expanded in a tree control, so that its children are displayed in the tree control.	
See also	<a href="#">&lt;tree-node&gt;</a> , page 607	

## **node-object** *Generic function*

Summary	Returns the object that the specified node in a tree control represents.	
Signature	<code>node-object tree-node =&gt; object</code>	

Arguments	<i>tree-node</i>	An instance of type < <b>tree-node</b> >.
Values	<i>object</i>	An instance of type < <b>object</b> >.
Description		Returns the object that <i>tree-node</i> represents.
See also		< <b>tree-node</b> >, page 607

## **node-parents** *Generic function*

Summary	Returns the parents of the specified node in a tree control.	
Signature	<b>node-parents</b> <i>tree-node</i> => <i>parents</i>	
Arguments	<i>tree-node</i>	An instance of type < <b>tree-node</b> >.
Values	<i>parents</i>	An instance of type < <b>sequence</b> >.
Description		Returns the parents of <i>tree-node</i> in a tree control.
See also	<b>node-children</b> , page 546 < <b>tree-node</b> >, page 607	

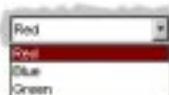
## **node-state** *Generic function*

Summary	Returns the state of the specified node in a tree control.	
Signature	<b>node-parents</b> <i>tree-node</i> => <i>state</i>	
Arguments	<i>tree-node</i>	An instance of type < <b>tree-node</b> >.
Values	<i>parents</i>	An instance of type <b>one-of</b> (#"expanded", #"contracted", #f).

Description	Returns the state of <i>tree-node</i> in a tree control, that is, whether it is currently expanded or contracted. This function returns #f if tree-node does not exist.
See also	<code>node-expanded?</code> , page 547 <code>&lt;tree-node&gt;</code> , page 607

## **<option-box>** *Open abstract instantiable class*

Summary	The class of option boxes.
Superclasses	<code>&lt;collection-gadget&gt;</code>
Init-keywords	<p><b>borders:</b> An instance of type <code>one-of(#f, #"none", #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output")</code>. Default value: #f.</p> <p><b>scroll-bars:</b> An instance of type <code>one-of(#f, #"none", #"horizontal", #"vertical", #"both", #"dynamic")</code>. Default value: #"both".</p>
Description	The class of option boxes.



The **borders:** init-keyword lets you specify a border around the option box. If specified, a border of the appropriate type is drawn around the gadget.

The **scroll-bars:** init-keyword lets you specify the scroll bar behavior for the gadget.

Internally, this class maps into the Windows drop-down list control.

Operations      None.

See also [<combo-box>](#), page 481

`<page>` *Open abstract instantiable class*

**Summary** The class that represents a page in a tab control.

## Superclasses <gadget>

**Init-keywords**    **label:**            An instance of type `type-union(<string>, <image>)`.

**Description** The class that represents a page in a multi-page frame, such as a tab control or wizard frame or property frame.

The `label`: init-keyword specifies a string or icon that is to be used as a label for the gadget. Pages typically appear inside a tab control, where the label for the page becomes the label on the tab for the page.

Operations dialog-next-page-setter dialog-previous-page-setter

See also [<property-page>](#), page 740

<tab-control-page>, page 582

<wizard-page>, page 754

## <password-field>

### *Open abstract instantiable class*

**Summary** The class of text fields that do not echo typed text.

Superclasses <text-field>

Init-keywords None.

Description	<p>The class of text fields that do not echo typed text. This class of gadgets are very similar in appearance to the <code>&lt;text-field&gt;</code> gadget, except that any text typed by the user is hidden in some way, rather than being echoed to the screen in the normal way.</p> <p>Internally, this class maps into the Windows single-line edit control with ES-PASSWORD style.</p>
Operations	None.
Example	<code>*pass* := contain(make(&lt;password-field&gt;));</code>
See also	<code>&lt;text-field&gt;</code> , page 593

## **<progress-bar>** *Open abstract instantiable class*

Summary	The class of progress bar windows.
Superclasses	<code>&lt;value-range-gadget&gt;</code>
Init-keywords	<code>orientation:</code> An instance of type <code>one-of(#"horizontal", #"<b>vertical</b>")</code> . Default value: <code>#"<b>horizontal</b>".</code>
Description	The class of progress bar windows.



The `orientation:` init-keyword lets you specify whether the progress bar should be horizontal or vertical.

Internally, this class maps into the Windows progress indicator control.

Operations      None.

Example      The following code creates an “empty” progress bar:

```
*prog* := contain
    (make(<progress-bar>,
          value-range:
            range(from: 0, to: 100)));
```

By setting the gadget value of the progress bar, the progress of a task can be monitored as follows:

```
for (i from 0 to 100) gadget-value(*prog*) := i end;
```

See also      <slider>, page 567

## <push-box>

*Open abstract instantiable class*

Summary      The class of grouped push buttons.

Superclasses      <button-box> <action-gadget>

Init-keywords      None.

Description      The class of grouped push buttons.



The **gadget-value** of a push box is always the gadget value of the last push button in the box to be pressed. You should use the gadget value of a push box as the way of determining which button has been pressed in a callback for the push box.

Operations      None.

**Example**

```
*push-box* := contain
  (make(<push-box>,
        items: range(from: 0, to: 5)));
```

**See also**

<check-box>, page 475  
 <group-box>, page 526  
 <radio-box>, page 556

**<push-button>***Open abstract instantiable class*

**Summary** The class of push buttons.

**Superclasses** <button> <action-gadget>

**Init-keywords** **default?:** An instance of type <boolean>. Default value: #f.

**Description** The class of push buttons. The push button gadget provides press-to-activate switch behavior.



When the button is activated (by releasing the pointer button over it), its activate callback is invoked.

If you supply a **gadget-value** for a push button, this can be used by any callback defined on the push button. This is especially useful in the case of push boxes, where this value can be used to test which button in the push box has been pressed.

The **default?:** init-keyword sets the default property for the push button gadget. When true, the push button is drawn with a heavy border, indicating that it is the “default operation” for that frame. Usually, this means that pressing the Return key invokes the activate callback.

Internally, this class maps into the push button Windows control.

Operations      None.

Example      The following code creates a push button which, when clicked, displays a message showing the label of the button.

```
contain(make(<push-button>,
            label: "Hello",
            activate-callback:
                method (gadget)
                    notify-user(format-to-string
                        ("Pressed button %=",
                         gadget-label(gadget)),
                        owner: gadget) end));
```

See also      [`<check-button>`](#), page 476

[`<radio-button>`](#), page 557

## **<push-menu-box>**

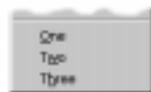
*Open abstract instantiable class*

Summary      The class of grouped push buttons in menus.

Superclasses      [`<menu-box>`](#) [`<action-gadget>`](#)

Init-keywords      None.

Description      The class of grouped push buttons in menus.



Internally, this class maps into the menu Windows control.

Operations      None.

Example      `contain(make(<push-menu-box>,  
                  items: range(from: 0, to: 5)));`

See also     [<check-menu-box>](#), page 477  
[<menu-box>](#), page 543  
[<radio-menu-box>](#), page 557

## <push-menu-button>

*Open abstract instantiable class*

Summary      The class of push buttons that appear on menus.

Superclasses    [<push-menu-button>](#)

Init-keywords    `default?:`      An instance of type [<boolean>](#). Default value: #f.

Description      The class of push buttons that appear on menus.



The `default?:` init-keyword sets the default value for the push menu button gadget.

Internally, this class maps into the menu item Windows control.

Operations    None.

See also     [<check-menu-button>](#), page 478  
[<menu-button>](#), page 544  
[<radio-menu-button>](#), page 558

<radio-box>		<i>Open abstract instantiable class</i>
Summary	The class of radio boxes, or groups of mutually exclusive radio buttons.	
Superclasses	<button-box> <action-gadget>	
Init-keywords	None.	
Description	The instantiable class that implements an abstract radio box, that is, a gadget that constrains a number of toggle buttons, only one of which may be selected at any one time.	
	The value of the radio box is the value of the currently selected item in the radio box.	
Operations	None.	
Examples	<pre>contain(make(&lt;radio-box&gt;, items: #("Yes", "No"),             orientation: #"vertical");</pre> <p>The following example defines a label-key function which formats the label of each item in the radio box, rather than just using the item itself.</p> <pre>*radio-box* := contain   (make(&lt;radio-box&gt;,         items: #(1, 2, 3, 4, 5),         orientation: #"vertical",         label-key:           method (item)             format-to-string("---%d---",                            item) end));</pre>	
See also	<p>&lt;check-box&gt;, page 475</p> <p>&lt;group-box&gt;, page 526</p> <p>&lt;push-box&gt;, page 552</p>	

**<radio-button>***Open abstract instantiable class*

**Summary** The class of radio buttons.

**Superclasses** `<button> <action-gadget>`

**Init-keywords** None.

**Description** The class of radio buttons. Isolated radio buttons are of limited use: you will normally want to combine several instances of such buttons using the `<radio-box>` gadget.



Internally, this class maps into the radio button Windows control.

**Operations** None.

**Example** `contain(make(<radio-button>, label: "Hello"));`

**See also** [<button>](#), page 470

[<check-button>](#), page 476

[<menu-button>](#), page 544

[<radio-box>](#), page 556

**<radio-menu-box>***Open abstract instantiable class*

**Summary** The class of grouped radio buttons that can appear in menus.

**Superclasses** `<menu-box> <action-gadget>`

**Init-keywords** None.

**Description** The class of grouped radio buttons that can appear in menus.

Internally, this class maps into the menu Windows control.

**Figure 8.12** A radio menu box

Operations	None.
Example	The following example creates a menu that shows an example of a radio menu box, as well as several other menu gadgets.
	<pre>contain(make(&lt;menu&gt;,             label: "Hello...",             children: vector                 (make(&lt;menu-button&gt;,                       label: "World"),                  make(&lt;menu-button&gt;,                       label: "Bonzo"),                  make(&lt;radio-menu-box&gt;,                       items:                           #("You", "All",                              "Everyone")),                  make(&lt;menu&gt;,                       label: "Others",                       children:                           vector                               (make(&lt;check-menu-box&gt;,                                     items: #(1, 2, 3)))                 )))); </pre>
See also	<p>&lt;menu-box&gt;, page 543</p> <p>&lt;push-menu-box&gt;, page 554</p> <p>&lt;radio-menu-button&gt;, page 558</p>

**<radio-menu-button>***Open abstract instantiable class*

Summary	The class of radio buttons that can appear in menus.
---------	--

Superclasses	<code>&lt;menu-button&gt;</code>
Init-keywords	None.
Description	<p>The class of radio buttons that can appear in menus. Isolated radio menu buttons are of limited use: you will normally want to combine several instances of such buttons using the <code>&lt;radio-menu-box&gt;</code> gadget.</p> <p>Internally, this class maps into the menu radio item Windows control.</p>
Operations	None.
Example	<code>contain(make(&lt;radio-menu-button&gt;, label: "Hello"));</code>
See also	<p><code>&lt;menu-button&gt;</code>, page 544</p> <p><code>&lt;push-menu-button&gt;</code>, page 555</p> <p><code>&lt;radio-menu-box&gt;</code>, page 557</p>



## remove-column *Generic function*

Summary	Removes a column from the specified table.	
Signature	<code>remove-column <i>table index</i> =&gt;</code>	
Arguments	<code>table</code>	An instance of type <code>&lt;table-control&gt;</code> .
	<code>index</code>	An instance of type <code>&lt;integer&gt;</code> .
Values	None.	
Description	Removes a column from <i>table</i> .	
See also	<code>add-column</code> , page 467	

**remove-item** *Generic function*

**Summary** Removes an item from a list control or table control.

**Signature** `remove-item list-or-table item => ()`

**Arguments** *list-or-table* An instance of `type-union(<list-control>, <table-control>)`.

*item* An instance of type `<list-item>`.

**Values** None

**Description** Removes *item* from *list-or-table*.

**See also** [add-item](#), page 467

[find-item](#), page 484

[<list-control>](#), page 531

[<list-item>](#), page 537

[make-item](#), page 538

[<table-control>](#), page 586

[<table-item>](#), page 592

**remove-node** *Generic function*

**Summary** Removes a node from a tree control.

**Signature** `remove-node tree node => ()`

**Arguments** *tree* An instance of `<tree-control>`.

*node* An instance of type `<tree-node>`.

**Values** None

Description	Removes <i>node</i> from <i>tree</i> .
See also	<a href="#">add-node</a> , page 468 <a href="#">find-node</a> , page 485 <a href="#">make-node</a> , page 540 <a href="#">&lt;tree-control&gt;</a> , page 598

## <scroll-bar> *Open abstract instantiable class*

Summary	The class of scroll bars.
Superclasses	<value-range-gadget>
Init-keywords	<p><b>orientation:</b> An instance of type <code>one-of(#"horizontal", "#vertical", #"none")</code>. Default value: <code>#"none"</code>.</p> <p><b>value-changing-callback:</b> An instance of type &lt;function&gt;.</p> <p><b>value-changed-callback:</b> An instance of type &lt;function&gt;.</p> <p><b>slug-size:</b> An instance of type &lt;real&gt;.</p>
Description	<p>The instantiable class that implements an abstract scroll bar.</p> <p>The <b>orientation:</b> init-keyword defines whether the scroll bar is horizontal or vertical.</p> <p>The <b>value-changing-callback:</b> init-keyword is the callback that is invoked when the gadget value is in the process of changing, such as when the scroll bar slug is dragged.</p> 

The **value-changed-callback:** init-keyword is the callback that is invoked when the gadget value has changed, such as when the scroll bar slug has come to rest after being dragged. You could use this callback, for example, to refresh the screen in your application to show a different part of a sheet, after the scroll bar had been moved.

The **slug-size:** init-keyword defines the size of the slug in the scroll bar, as a proportion of **value-range:**. For example, if **value-range:** is from 0 to 100, and **slug-size:** is 25, then the slug occupies a quarter of the total length of the scroll bar. The slug is the part of the scroll bar that can be dragged up and down, and represents how much of the sheet being scrolled is visible.

**Note:** The Microsoft Windows Interface Guidelines refer to the slug as a *scroll-box*, and the area in which the slug can slide as the *scroll-shaft*. You should be aware of this difference if you are using those guidelines as a reference.

Internally, this class maps into the Windows scroll bar control.

Operations      **gadget-slug-size** **gadget-slug-size-setter**

Example      As an example of how the **slug-size:** init-keyword operates, compare the two examples of scroll bars below. The second scroll bar has a slug that is twice the size of the first.

```
contain(make(<scroll-bar>,
            value-range: range(from: 0, to: 100)
            slug-size: 10));

contain(make(<scroll-bar>,
            value-range: range(from: 0, to: 100)
            slug-size: 20));
```

See also      [\*\*<slider>\*\*](#), page 567

scrolling	<i>Statement macro</i>				
Summary	Places scroll bars around the specified DUIM panes, if required.				
Macro call	<code>scrolling ([<i>options</i>]) {<i>pane</i>} end</code>				
Arguments	<table> <tr> <td><i>options</i></td><td>Dylan arguments<sub>bnf</sub>.</td></tr> <tr> <td><i>pane</i></td><td>A Dylan expression<sub>bnf</sub>.</td></tr> </table>	<i>options</i>	Dylan arguments <sub>bnf</sub> .	<i>pane</i>	A Dylan expression <sub>bnf</sub> .
<i>options</i>	Dylan arguments <sub>bnf</sub> .				
<i>pane</i>	A Dylan expression <sub>bnf</sub> .				
Values	None.				
Description	<p>Places scroll bars around the DUIM panes created by <i>pane</i>, if required. It is useful to use this macro if you are unsure that the panes created can be displayed on the screen successfully without scroll bars: this macro only adds scroll bars when it is necessary.</p> <p>Creates <i>pane</i> with scroll bars attached to it, taking into account any of the specified <i>options</i>.</p> <p>The <i>pane</i> is an expression whose return value is the sheet to which the scroll bars should be attached.</p> <p>The options can be used to specify the properties of the scroll bars. As well as all the properties of &lt;gadget&gt;, these include a <b>scroll-bars:</b> init-keyword, which may take one of the following values: #f, #"none", #"horizontal", #"vertical", #"both", #"dynamic". If no options are specified, then both vertical and horizontal scroll bars are used.</p> <p>The pane is a body of code whose return value is the sheet to which the label should be assigned.</p>				
Example	<pre>scrolling (scroll-bars: #"vertical")   make(&lt;radio-box&gt;,     orientation: #"vertical",     items: range(from: 1, to: 50)) end</pre>				

See also      [`<scroll-bar>`](#), page 561  
[`scroll-position`](#), page 564  
[`set-scroll-position`](#), page 565

## **scroll-position** *Generic function*

Summary      Returns the position of the scroll bar slug in the specified sheet.

Signature      `scroll-position sheet => xy`

Arguments      `sheet`      An instance of type `<sheet>`.

Values      `x`      An instance of type `<integer>`.  
`y`      An instance of type `<integer>`.

Description      Returns the position of the scroll bar slug in `sheet`. Note that this generic function only returns the position of scroll bar slugs that have been created using the `scrolling` macro. It does not work on gadgets with scroll bars defined explicitly.  
  
**Note:** The Microsoft Windows Interface Guidelines refer to the slug as a *scroll-box*, and the area in which the slug can slide as the *scroll-shaft*. You should be aware of this difference if you are using those guidelines as a reference.

### Example

See also      [`scrolling`](#), page 563  
[`set-scroll-position`](#), page 565

## **<separator>** *Open abstract instantiable class*

Summary      The class of gadgets used as a visual separator.

Superclasses `<gadget>`

Init-keywords `orientation:` An instance of type `one-of(#"horizontal", #"vertical"). Default value: #"horizontal".`

Description The class of gadgets used as a visual separator.



**Figure 8.13** A separator

The `orientation:` init-keyword specifies whether the separator is vertical or horizontal.

Operations None.

Example The following example creates a column layout and places two buttons in it, separated with a separator.

```
contain(vertically ()
    make(<button>, label: "Hello");
    make(<separator>);
    make(<button>, label: "World")
end);
```

See also `<spacing>`, page 569

## set-scroll-position

*Generic function*

Summary Scrolls the window on the specified sheet.

Signature `set-scroll-position sheet x y => ()`

Arguments `sheet` An instance of type `<sheet>`.

`x` An instance of type `<integer>`.

	<b>y</b>	An instance of type <integer>.
Values	None.	
Description	Scrolls the window on <i>sheet</i> by setting the position of the scroll bar slug. Note that this generic function only sets the position of scroll bar slugs that have been created using the <code>scrolling</code> macro. It does not work on gadgets with scroll bars defined explicitly.	
		<b>Note:</b> The Microsoft Windows Interface Guidelines refer to the slug as a <i>scroll-box</i> , and the area in which the slug can slide as the <i>scroll-shaft</i> . You should be aware of this difference if you are using those guidelines as a reference.
See also	<code>scroll-position</code> , page 564 <code>scrolling</code> , page 563	

<b>sheet-viewport</b>		<i>Generic function</i>
Summary		Returns the viewport that is clipping the specified sheet.
Signature		<code>sheet-viewport sheet =&gt; viewport</code>
Arguments	<i>sheet</i>	An instance of type <sheet>.
Values	<i>viewport</i>	An instance of type <code>false-or(&lt;viewport&gt;)</code> .
Description		Returns the viewport that is clipping <i>sheet</i> .
See also		<code>sheet-viewport-region</code> , page 567 <code>&lt;viewport&gt;</code> , page 610

## **sheet-viewport-region** *Generic function*

Summary	Returns the sheet region of the specified sheet's viewport, if it has one.
Signature	<code>sheet-viewport-region sheet =&gt; region</code>
Arguments	<code>sheet</code> An instance of type <code>&lt;sheet&gt;</code> .
Values	<code>region</code> An instance of type <code>&lt;region&gt;</code> .
Description	Returns the sheet region of <code>sheet</code> 's viewport, if it has one. If <code>sheet</code> has no viewport, it returns <code>sheet</code> 's own region.
See also	<code>sheet-viewport</code> , page 566 <code>&lt;viewport&gt;</code> , page 610

## **<slider>** *Open abstract instantiable class*

Summary	The class of slider gadgets.
Superclasses	<code>&lt;value-range-gadget&gt;</code>
Init-keywords	<p><code>min-label:</code> An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code>.</p> <p><code>max-label:</code> An instance of type <code>type-union(&lt;string&gt;, &lt;image&gt;)</code>.</p> <p><code>borders:</code> An instance of type <code>one-of(#f, #"none", #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output")</code>. Default value: <code>#f</code>.</p> <p><code>tick-marks:</code> An instance of type <code>false-or(&lt;integer&gt;)</code>. Default value: <code>#f</code></p>

**orientation:** An instance of type `one-of(#"horizontal", "#"vertical")`. Default value: `#"horizontal"`.

**value-changing-callback:**

An instance of type `<function>`.

Description	The class of slider gadgets. This is a gadget used for setting or adjusting the value on a continuous range of values, such as a volume or brightness control.	
-------------	--	--

You can specify a number of attributes for the labels in a slider. The `min-label:` and `max-label:` init-keywords let you specify a label to be displayed at the minimum and maximum points of the slider bar, respectively. In addition, the `range-label-text-style:` init-keyword lets you specify a text style for these labels.

The `borders:` init-keyword lets you specify a border around the slider. If specified, a border of the appropriate type is drawn around the gadget.

The `tick-marks:` init-keyword specifies the number of tick-marks that are shown on the slider. Displaying tick marks gives the user a better notion of the position of the slug at any time.

The `orientation:` init-keyword specifies whether the slider is horizontal or vertical.

The `value-changing-callback:` init-keyword is the callback that is invoked when the slider slug is dragged.

Internally, this class maps into the Windows trackbar control.

When designing a user interface, you will find that spin boxes are a suitable alternative to spin boxes in many situations.

Operations	None.
Example	<pre>contain(make(&lt;slider&gt;,             value-range:                 range(from: -20, to: 20, by: 5)));</pre>
See also	<a href="#">&lt;scroll-bar&gt;</a> , page 561 <a href="#">&lt;spin-box&gt;</a> , page 570

## <spacing> *Open abstract instantiable class*

Summary	The class of gadgets that can be used to provide spacing around a sheet.
Superclasses	<a href="#">&lt;gadget&gt;</a>
Init-keywords	<p><b>child:</b> An instance of type <code>limited(&lt;sequence&gt; of: &lt;sheet&gt;)</code>.</p> <p><b>thickness:</b> An instance of type <a href="#"><code>&lt;integer&gt;</code></a>. Default value: 1.</p>
Description	<p>The class of gadgets that can be used to provide spacing around a sheet.</p> <p>The <b>child:</b> init-keyword is the sheet or sheets that you are adding spacing around.</p> <p>The <b>thickness:</b> init-keyword specifies the thickness of the spacing required.</p> <p>It is usually clearer to use the <code>with-spacing</code> macro, rather than to create an instance of <a href="#"><code>&lt;spacing&gt;</code></a> explicitly.</p>
Operations	None.
Example	The following creates a vertical layout containing two buttons separated by a text field that has spacing added to it.

```

contain(vertically ()
    make(<button>, label: "Hello");
    make(<spacing>,
        child: make(<text-field>),
        thickness: 10);
    make(<button>, label: "World")
end);

```

See also      [<null-pane>](#), page 420  
[<separator>](#), page 564  
[with-spacing](#), page 613

## <spin-box>

*Open abstract instantiable class*

Summary      The class of spin box gadgets.

Superclasses      [<collection-gadget>](#)

Init-keywords      **borders:**      An instance of type one-of(#f, #"none", #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output"). Default value: #f.

Description      The class of spin box gadgets. A spin box gadget is a text box that only accepts a limited range of values that form an ordered loop. As well as typing a value directly into the text box, small buttons are placed on its right hand side (usually with up and down arrow icons as labels). You can use these buttons to increase or decrease the value shown in the text box.



A spin box may be used when setting a percentage value, for example. In this case, only the values between 0 and 100 are valid, and a spin box is a suitable way of ensuring that only valid values are specified by the user.

The **borders**: init-keyword lets you specify a border around the spin box. If specified, a border of the appropriate type is drawn around the gadget.

When designing a user interface, you will find that sliders are a suitable alternative to spin boxes in many situations.

Operations

None.

Example

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

See also

[<slider>](#), page 567

## <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>`.

<b>splitter-split-box-callback-setter</b>	<i>Generic function</i>
---	-------------------------

```
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>`.

<b>&lt;status-bar&gt;</b>	<i>Open abstract instantiable class</i>
---------------------------	---

**Summary**      The class of status bars.

**Superclasses**    `<value-range-gadget>`

**Init-keywords**    **label:**      An instance of type `type-union(<string>, <image>)`.

**label-pane:**    An instance of `false-or(<gadget>)`.  
                      Default value: `#f`.

**progress-bar?:** An instance of type `<boolean>`. Default  
                      value: `#f`.

**progress-bar:** An instance of `false-or(<progress-bar>)`.  
                      Default value: `#f`.

**value:**        An instance of type `<object>`.

**value-range:** An instance of type `<range>`.

**Description**      The class of status bars. Status bars are often used at the bottom of an application window, and can provide a range of feedback on the current state of an application. Some examples of information that is often placed in a status bar are:

- Documentation strings for the currently selected menu button.

- Progress indicators to show the state of operations such as loading and saving files.
- The current position of the caret on the screen.
- Currently selected configurable values (such as the current font family, size, and style in a word processor).
- The current time.



**Figure 8.14** A status bar

In particular, it is trivial to add an in-built progress bar to a status bar. Any documentation strings specified for menu buttons in a frame are automatically displayed in the label pane of a status bar when the mouse pointer is placed over the menu button itself.

The `label:` init-keyword specifies a string or icon that is to be used as a label for the gadget. Alternatively, the `label-pane:` init-keyword specifies a pane that should be used as the label. You should only use one of these init-keywords; see the discussion about creating status bars below.

If `progress-bar?:` is true, then the status bar has a progress bar. Alternatively, the `progress-bar:` init-keyword specifies a pane that should be used as the label. You should only use one of these init-keywords; see the discussion about creating status bars below.

The `value:` init-keyword specifies the gadget value of the progress bar, if there is one.

The `value-range:` init-keyword is the range of values across which the progress bar can vary, if there is one.

Internally, this class maps into the Windows status window control.

There are two ways that you can create a status bar:

- The simple way is to provide a simple status bar that only has a label and, optionally, a progress bar.
- The more complicated way is to define all the elements of a status bar from scratch, as children of the status bar.

If you want to create a simple status bar, then use the `label: init`-keyword to specify the text to be displayed in the status bar. In addition, you can set or check the label using `gadget-label` once the status bar has been created.

You can create a basic progress bar by setting `progress-bar?: true`. If you create a progress bar in this way, then it will respond to the `gadget-value` and `gadget-value-range` protocols: you can use `gadget-value` to set the position of the progress bar explicitly, or to check it, and you can use `gadget-value-range` to define the range of values that the progress bar can take, just like any other value gadget. By default, the range of possible values is 0 to 100.

The more complicated way to create a status bar is to define all its children from scratch. You need to do this if you want to provide the user with miscellaneous feedback about the application state, such as online documentation for menu commands, or the current position of the cursor. Generally speaking, if you need to provide pane in which to display information, you should define instances of `<label>` for each piece of information you want to use. However, if you wish you can add any type of gadget to your status bar in order to create a more interactive status bar. For instance, many word processors include gadgets in the status bar that let you select the zoom level at which to view the current document from a drop-down list of options.

If you define the children of a status bar from scratch in this way, you should make appropriate use of the `label-pane:` and `progress-bar?:` init-keywords. The `label-pane:` init-keyword lets you specify the pane that is to act as the label for the status bar; that is, the pane that responds to the `gadget-label` protocol. The `progress-bar?:` init-keyword lets you define a progress bar to add to the status bar. If you create a status bar from scratch, you should not use either the `label:` or `progress-bar?:` init-keywords.

Operations	<code>frame-status-bar-setter make status-bar-label-pane status-bar-progress-bar</code>
Example	The following creates a basic status bar with the given label, and a progress bar with the given range of values.
	<pre>contain(make(&lt;status-bar&gt;,             progress-bar?: #t,             value-range: range(from: 0, to: 50)             label: "Status"));</pre>

See also	<code>frame-status-bar</code> , page 719 <code>frame-status-message</code> , page 720 <code>gadget-documentation</code> , page 495 <code>status-bar-label-pane</code> , page 576 <code>status-bar-progress-bar</code> , page 577
----------	--

## status-bar-label-pane

*Generic function*

Summary	Returns the gadget that displays the label of the specified status bar.
Signature	<code>status-bar-label-pane status-bar =&gt; label</code>
Arguments	<code>status-bar</code> An instance of type <code>&lt;status-bar&gt;</code> .

Values *label* An instance of type `false-or(<label>)`.

Description Returns the gadget that displays the label of *status-bar*.

Example Create a status bar with a label as follows:

```
*status* := contain(make(<status-bar>,
                         value-range:
                           range(from: 0, to: 100),
                         label: "Status"));
```

The pane that the label of the status bar is displayed in can be returned with the following call:

```
status-bar-label-pane(*status*);
```

See also `<status-bar>`, page 573

`status-bar-progress-bar`, page 577

## **status-bar-progress-bar** *Generic function*

Summary Returns the progress bar for the specified status bar.

Signature `status-bar-progress-bar status-bar => progress-bar`

Arguments *status-bar* An instance of type `<status-bar>`.

Values *progress-bar* An instance of type `false-or(<progress-bar>)`.

Description Returns the progress bar for *status-bar*, if there is one.

See also `<progress-bar>`, page 551

## **<tab-control>** *Open abstract instantiable class*

Summary The class of tab controls.

Superclasses	<code>&lt;value-gadget&gt;</code>
Init-keywords	<p><code>pages:</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;page&gt;)</code>.</p> <p><code>current-page:</code> An instance of type <code>false-or(&lt;sheet&gt;)</code>.</p> <p><code>key-press-callback:</code> An instance of type <code>false-or(&lt;command&gt;, &lt;function&gt;)</code>.</p>
Description	<p>The class of tab controls. These controls let you implement a multi-page environment in a window or dialog. Each page in a tab control has its own associated layout of sheets and gadgets, and an accompanying tab (usually displayed at the top of the page, rather like the tab dividers commonly used in a filing cabinet). Each page in a tab control can be displayed by clicking on the appropriate tab.</p> <p>The <code>pages:</code> init-keyword is used to define the pages that the tab control contains. Each page in the tab control is an instance of the class <code>&lt;page&gt;</code>.</p> <p>The <code>current-page:</code> init-keyword specifies which tab is visible when the tab control is first displayed.</p> <p>The <code>key-press-callback:</code> init-keyword lets you specify a key-press callback. This type of callback is invoked whenever a key on the keyboard is pressed while the gadget has focus. In a tab control, a key-press callback might be used as a quick way to display each page in the tab control. See <code>gadget-key-press-callback</code>, page 502, for a fuller description of key-press callbacks.</p> <p>The <code>gadget-id</code> of a tab control is particularly useful, because it is returned by <code>gadget-value</code>.</p> <p>Internally, this class maps into the Windows tab control.</p>



**Operations** `tab-control-current-page tab-control-current-page-setter tab-control-labels tab-control-pages tab-control-pages-setter`

**Example** The following example creates a tab control that has two pages. The first page contains a button, and the second page contains a list.

```
contain(make(<tab-control>,
    pages:
        vector(make(<tab-control-page>,
            label: "First",
            child: make(<push-button>,
                label: "One")),
            make(<tab-control-page>,
                label: "Second",
                child: make(<list-box>,
                    items:
                        #(1, 2, 3)
                ))));
});
```

See also `<page>`, page 550

## **tab-control-current-page** *Generic function*

**Summary** Returns the current visible page of the specified tab control.

**Signature** `tab-control-current-page tab-control => visible-page`

**Arguments** `tab-control` An instance of type `<tab-control>`.

**Values** `visible-page` An instance of type `<page>`.

**Description** Returns the current visible page of `tab-control`.

**Example** The following example creates a tab control that has two pages.

```
*tab* := contain
  (make
    (<tab-control>,
     pages:
      vector(make(<tab-control-page>,
                  label: "First",
                  child: make(<push-button>,
                              label: "One")),
            make(<tab-control-page>,
                  label: "Second",
                  child: make(<list-box>,
                              items:
                                #(1, 2, 3)
                ))));

```

The current page of the tab control can be returned with the following code:

```
tab-control-current-page(*tab*);
```

**See also**

<page>, page 550

<tab-control>, page 577

tab-control-current-page-setter, page 580

tab-control-pages, page 583

## tab-control-current-page-setter

*Generic function*

**Summary** Sets the current visible page of the specified tab control.

**Signature** `tab-control-current-page-setter visible-page tab-control => visible-page`

**Arguments** `visible-page` An instance of type <page>.

`tab-control` An instance of type <tab-control>.

**Values** `visible-page` An instance of type <page>.

Description Sets the current visible page of *tab-control*.

Example The following example creates a tab control that has two pages.

```
*tab* := contain
  (make
    (<tab-control>,
     pages:
      vector(make(<tab-control-page>,
                  label: "First",
                  child: make(<push-button>,
                              label: "One")),
            make(<tab-control-page>,
                  label: "Second",
                  child: make(<list-box>,
                              items:
                                #(1, 2, 3)
                  ))));

```

Assign a variable to the current page of the tab control as follows:

```
*page* := tab-control-current-page(*tab*);
```

Next, change the current page of the tab control by clicking on the tab for the hidden page. Then, set the current page to be the original current page as follows:

```
tab-control-current-page(*tab*) := *page*;
```

See also [<page>](#), page 550

[<tab-control>](#), page 577

[tab-control-current-page](#), page 579

## tab-control-labels

*Generic function*

Summary Returns the tab labels of the specified pane.

Signature `tab-control-labels tab-control => labels`

Arguments	<i>tab-control</i>	An instance of type <code>&lt;tab-control&gt;</code> .
Values	<i>labels</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;label&gt;)</code> .
Description	Returns the tab labels of <i>tab-control</i> , as a sequence. Each element in <i>labels</i> is an instance of <code>&lt;label&gt;</code> .	
Example	Given the tab control created by the code below:	

```
*tab* := contain
    (make
        (<tab-control>,
         pages:
            vector(make(<tab-control-page>,
                         label: "First"),
                  make(<tab-control-page>,
                         label: "Second"),
                  make(<tab-control-page>,
                         label: "Third"),
                  make(<tab-control-page>,
                         label: "Fourth"),
                  make(<tab-control-page>,
                         label: "Fifth"))));
```

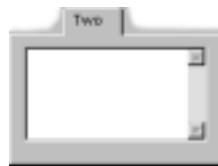
You can return a list of the labels as follows:

```
tab-control-labels(*tab*);
```

See also	<code>&lt;tab-control&gt;</code> , page 577 <code>tab-control-pages</code> , page 583
----------	--

<b>&lt;tab-control-page&gt;</b>		<i>Open abstract instantiable class</i>
Summary	The class that represents a page in a tab control.	
Superclasses	<code>&lt;page&gt;</code>	
Init-keywords	None.	

Description The class that represents a page in a tab control.



**Figure 8.15** A tab control page

Operations None.

See also [`<page>`](#), page 550  
[`<tab-control>`](#), page 577  
[`<property-page>`](#), page 740  
[`<wizard-page>`](#), page 754

## tab-control-pages

## *Generic function*

Summary Returns the tab pages of the specified pane.

Signature `tab-control-pages tab-control => pages`

Arguments `tab-control` An instance of type [`<tab-control>`](#).

Values `pages` An instance of type `limited(<sequence>, of: <page>)`. Default value: `#[]`.

Description Returns the tab pages of *pane*.

**Example**

Given the tab control created by the code below:

```
*tab* := contain
    (make
        (<tab-control>,
        pages:
            vector(make(<tab-control-page>,
                label: "First"),
                make(<tab-control-page>,
                    label: "Second"),
                make(<tab-control-page>,
                    label: "Third"),
                make(<tab-control-page>,
                    label: "Fourth"),
                make(<tab-control-page>,
                    label: "Fifth"))));
```

You can return a list of the pages as follows:

```
tab-control-pages(*tab*);
```

**See also**

<page>, page 550  
<tab-control>, page 577  
tab-control-current-page, page 579  
tab-control-labels, page 581  
tab-control-pages-setter, page 584

**tab-control-pages-setter**

*Generic function*

**Summary**

Sets the tab pages of the specified tab control.

**Signature**

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

**Arguments**

<i>pages</i>	An instance of type limited(<sequence>, of: <page>).
<i>tab-control</i>	An instance of <tab-control>.
<i>page</i>	An instance of <page>.

Values	<code>pages</code>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;page&gt;)</code> .
Description		Sets the tab pages available to <i>tab-control</i> , optionally setting <i>page</i> to the default page to be displayed. The <i>pages</i> argument is an instance of <code>limited(&lt;sequence&gt;, of: &lt;page&gt;)</code> . The <i>page</i> argument is an instance of <code>&lt;page&gt;</code> and, moreover, must be one of the pages contained in <i>pages</i> .
The <code>tab-control-pages-setter</code> function is used as follows:		
<pre>tab-control-pages(my-tab-control, page: my-page) := my-pages</pre>		
See also	<a href="#">&lt;page&gt;</a> , page 550 <a href="#">&lt;tab-control&gt;</a> , page 577 <a href="#">tab-control-pages</a> , page 583	

## **<table-column>** *Sealed class*

Summary	The class of columns in table controls.										
Superclasses	<code>&lt;object&gt;</code>										
Init-keywords	<table> <tr> <td><code>heading:</code></td> <td>An instance of type <code>&lt;string&gt;</code>.</td> </tr> <tr> <td><code>width:</code></td> <td>An instance of type <code>&lt;integer&gt;</code>. Default value: 100.</td> </tr> <tr> <td><code>alignment:</code></td> <td>An instance of type <code>one-of(#"left", #"right", #"center")</code>. Default value: <code>#"left"</code>.</td> </tr> <tr> <td><code>generator:</code></td> <td>An instance of type <code>&lt;function&gt;</code>.</td> </tr> <tr> <td><code>callback:</code></td> <td>An instance of type <code>false-or(&lt;function&gt;)</code>. Default value: <code>#f</code>.</td> </tr> </table>	<code>heading:</code>	An instance of type <code>&lt;string&gt;</code> .	<code>width:</code>	An instance of type <code>&lt;integer&gt;</code> . Default value: 100.	<code>alignment:</code>	An instance of type <code>one-of(#"left", #"right", #"center")</code> . Default value: <code>#"left"</code> .	<code>generator:</code>	An instance of type <code>&lt;function&gt;</code> .	<code>callback:</code>	An instance of type <code>false-or(&lt;function&gt;)</code> . Default value: <code>#f</code> .
<code>heading:</code>	An instance of type <code>&lt;string&gt;</code> .										
<code>width:</code>	An instance of type <code>&lt;integer&gt;</code> . Default value: 100.										
<code>alignment:</code>	An instance of type <code>one-of(#"left", #"right", #"center")</code> . Default value: <code>#"left"</code> .										
<code>generator:</code>	An instance of type <code>&lt;function&gt;</code> .										
<code>callback:</code>	An instance of type <code>false-or(&lt;function&gt;)</code> . Default value: <code>#f</code> .										
Description	The class of columns in table controls.										

The `width:` init-keyword lets you specify the width of the column. The `alignment:` init-keyword is used to specify how the column should be aligned in the table.

To populate the table column, the function specified by `generator:` is invoked. This function is called for each item in the table control, and the value returned is placed at the appropriate place in the column.

In addition, you can also specify a callback that can be used for sorting the items in the table column, using the `callback:` init-keyword.

Operations      None.

Example

See also      `<table-control>`, page 586

## **<table-control>**

*Open abstract instantiable class*

Summary      The class of table controls.

Superclasses    `<collection-gadget>` `<action-gadget>`

Init-keywords	<code>headings:</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;string&gt;)</code> .  <code>generators:</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;function&gt;)</code> .  <code>view:</code> An instance of type <code>&lt;table-control-view&gt;</code> . Default value: <code>#"table"</code> .  <code>borders:</code> An instance of type <code>one-of(#f, #"none", #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output")</code> . Default value: <code>#f</code> .
---------------	--

**scroll-bars:** An instance of type `one-of(#f, #"none",  
#"horizontal", #"vertical", #"both",  
#"dynamic")`. Default value: `#"both"`.

**popup-menu-callback:**

An instance of type `<function>`.

**key-press-callback:**

An instance of type `false-or(<command>,  
<function>)`.

**widths:** An instance of type `limited(<sequence>,  
of: <integer>)`.

Description The class of table controls.

Squared	Doubled
1	2
4	4
9	6
16	8
25	10

The `view:` init-keyword can be used to specify how the items in the table control are displayed. See `<table-control-view>`, page 589, for more details.

The `borders:` init-keyword lets you specify a border around the table control. If specified, a border of the appropriate type is drawn around the gadget.

The `scroll-bars:` init-keyword defined the scroll bar behavior for the gadget.

You can use the `popup-menu-callback:` init-keyword to specify a context-sensitive menu to display for one or more selected items in the table control. In Windows 95, for instance, such a context-sensitive menu can be displayed by right-clicking on any item or group of selected items in the list control.

The `key-press-callback`: init-keyword lets you specify a key-press callback. This type of callback is invoked whenever a key on the keyboard is pressed while the gadget has focus. In a table control, a key-press callback might be used as a quick way to select an item in the control. See `gadget-key-press-callback`, page 502, for a fuller description of key-press callbacks.

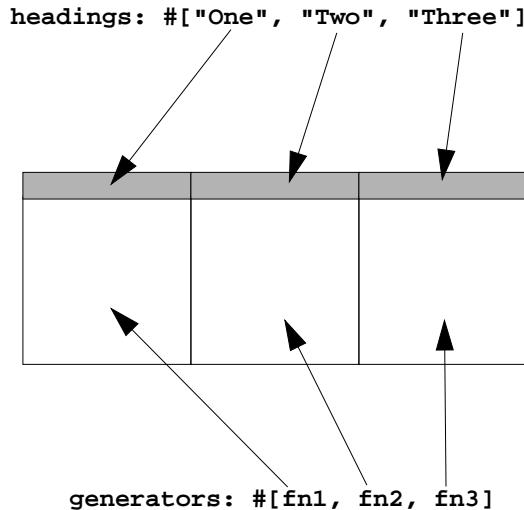
The `headings:` and `generators:` init-keywords can be used to specify the titles of each column in the control, and a sequence of functions that are used to generate the contents of each column. The headings should be a sequence of strings, and the generators should be a sequence of functions.

The first item in the sequence of headings is used as the title for the first column, the second is used as the title of the second column, and so on. Similarly, the first function in the sequence of generators is invoked on each item in the control, thereby generating the contents of the first column, the second is used to generate the contents of the second column by invoking it on each item in the control, and so on. This is illustrated in Figure 8.16.

If you do not specify both of these init-keywords, you must supply columns for the table control, using the `<table-column>` class.

The `widths:` init-keyword lets you specify the width of each column in the table control. It takes a sequence of integers, each of which represents the width, in pixels, of the respective column in the control. Note that there must be as many widths as there are columns.

Internally, this class maps into the Windows list view control with LVS-REPORT style.



**Figure 8.16** Defining column headings and contents in table controls

Operations      `add-column` `remove-column` `table-control-view` `table-control-view-setter`

See also      `<table-column>`, page 585  
`<table-control-view>`, page 589

### **<table-control-view>** *Type*

Summary      The type of possible views for a table control

Equivalent      `one-of(#"table", #"small-icon", #"large-icon", #"list")`

Superclasses      None.

Init-keywords      None.

Description	This type represents the acceptable values for the view arguments to operators of < <b>table-control</b> >. There are four possible values, corresponding to the view options that will be familiar to most users of GUI-based operating systems:
<b>#"small-icon"</b>	Displays each item in the table with a small icon to the left of the item. Items are arranged horizontally.
<b>#"large-icon"</b>	Displays each item in the table with a large icon to the left of the item. Items are arranged horizontally.
<b>#"list"</b>	Displays each item in the table with a small icon to the left of the item. Items are arranged vertically in one column.
<b>#"table"</b>	Displays each item in the list with a small icon to the left of the item. Items are arranged vertically in one column. Additional details not available in other views are also displayed. The details that are displayed depend on the nature of the items in the table control. For example, if filenames are displayed in the table control, additional details may include the size, modification date, and creation date of each file. If e-mail messages are displayed in the table control, additional details may include the author of the e-mail, its subject, and the date and time it was sent.
See also	<a href="#"><b>&lt;list-control-view&gt;</b></a> , page 534 <a href="#"><b>&lt;table-control&gt;</b></a> , page 586 <a href="#"><b>table-control-view</b></a> , page 591

## table-control-view

### *Generic function*

Summary	Returns the current view of the specified table control.
Signature	<code>table-control-view table-control =&gt; view</code>
Arguments	<i>table-control</i> An instance of type < <code>table-control</code> >.
Values	<i>view</i> An instance of type < <code>table-control-view</code> >.
Description	Returns the current view of <i>table-control</i> . The available views are described in the entry for < <code>table-control-view</code> >, page 589.
See also	<a href="#">&lt;table-control-view&gt;</a> , page 589 <a href="#">table-control-view-setter</a> , page 591

## table-control-view-setter

### *Generic function*

Summary	Sets the current view of the specified table control.
Signature	<code>table-control-view-setter view table-control =&gt; view</code>
Arguments	<i>view</i> An instance of type < <code>table-control-view</code> >. <i>table-control</i> An instance of type < <code>table-control</code> >.
Values	<i>view</i> An instance of type < <code>table-control-view</code> >.
Description	<p>Sets the current view of <i>table-control</i>.</p> <p>The <i>view</i> argument is used to specify the way in which the items in the table control are displayed.</p>
See also	<a href="#">&lt;table-control-view&gt;</a> , page 589 <a href="#">table-control-view</a> , page 591

**<table-item>** *Open abstract instantiable class*

Summary The class that represents an item in a table control.

Superclasses `<object>`

Init-keywords `object:` An instance of type `<object>`.

Description The class that represents an item in a table control.

The `object:` init-keyword describes the object that an instance of table item represents.

Operations None.

See also [add-item](#), page 467

[find-item](#), page 484

[make-item](#), page 538

[remove-item](#), page 560

[<table-control>](#), page 586

**<text-editor>** *Open abstract instantiable class*

Summary The class of multiple line text editors.

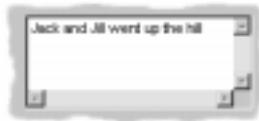
Superclasses `<text-field>`

Init-keywords `columns:` An instance of type `false-or(<integer>)`.  
Default value: `#f`.

`lines:` An instance of type `false-or(<integer>)`.  
Default value: `#f`.

`scroll-bars:` An instance of type `one-of(#f, #"none", #"horizontal", #"vertical", #"both", #"dynamic")`. Default value: `#"both"`.

Description The class of multiple line text editors.



The `columns:` and `lines:` init-keywords specify the number of columns and lines of characters visible in the text editor, respectively.

The `scroll-bars:` init-keyword specifies whether the text editor has scroll bars or not.

Internally, this class maps into the multi-line edit control Windows control.

Operations None.

Examples To constrain the number of lines and columns when an editor is first displayed:

```
*editor* := contain(make(<text-editor>,
                      lines: 20, columns: 80));
```

To make a text editor that is fixed at 10 lines high:

```
make(<text-editor>, lines: 10, fixed-height?: #t);
```

See also [<text-field>](#), page 593

## **<text-field>** *Open abstract instantiable class*

Summary The class of single line text fields.

Superclasses [<text-gadget>](#)

Init-keywords	<p><b>x-alignment:</b> An instance of type <code>one-of(#"left", # "right", # "center")</code>. Default value: <code>#"left"</code>.</p> <p><b>case:</b> An instance of type <code>one-of(#f, # "lower", # "upper")</code>. Default value: <code>#f</code>.</p> <p><b>auto-scroll?:</b> An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#f</code>.</p>
---------------	---

Description	The class of single line text fields.
-------------	---------------------------------------



The **x-alignment:** init-keyword is used to align the text in the text field.

The **case:** init-keyword lets you specify which case is used to display the text in the text field. You can specify either upper or lower case. The default is to display letters of either case.

If **auto-scroll?:** is true, then text scrolls off to the left of the text field if more text is typed than can be displayed in the text field itself.

Internally, this class maps into the single-line edit control Windows control.

Operations	None.
------------	-------

Example	To make a text field with a fixed width:
---------	--

```
make(<text-field>, width: 200, fixed-width?: #t);
```

The following example creates a text field which, after pressing Return, invokes a callback that displays the gadget value in a dialog box.

```
*text* := contain
  (make(<text-field>,
        value-changed-callback:
          method (gadget)
            notify-user
              (format-to-string
                ("Changed to %=",
                  gadget-value(gadget)),
                owner: gadget) end));
```

See also      [<password-field>](#), page 550

## <text-gadget> *Open abstract class*

Summary      The class of all text gadgets.

Superclasses    <value-gadget> <action-gadget>

Init-keywords    **text:**      An instance of type <string>. Default value:  
 "".

**value-type:**    An instance of type <type>. Default value:  
 <string>.

**value-changing-callback:**  
 An instance of type **false-or(<function>)**.

Description      The class of all text gadgets. You should not create a direct  
 instance of this class.

The **text:** init-keyword specifies a text value for the combo  
 box.

The **value-type:** init-keyword specifies the type of the  
 gadget value of the text gadget, which by default is  
 <string>. Other supported types are <integer> and <sym-  
 bol>. The string entered in the text gadget is parsed, and con-  
 verted to the appropriate type automatically.

Text gadgets have a method `on gadget-value` that converts the `gadget-text` based on the `gadget-value-type`, for example converting the string to an integer for `value-type: <integer>`.

The `gadget-text` generic function always returns the exact text typed into a text gadget. However, `gadget-value` always returns a “parsed” value of the appropriate type, depending on the value of `gadget-value-type`. If the string contains any characters that are not appropriate to the `gadget-value-type` (for example, if the string contains any non-integers, and the `gadget-value-type` is `<integer>`), then `gadget-value` returns `#f`.

Setting the gadget value “prints” the value and inserts the appropriate text into the text field.

The `value-changingCallback: init`-keyword allows you to specify a callback that is invoked as the value of the text gadget is changing during the course of “casual” typing. Generally, this means when the user is typing text, but before the text is committed (usually by pressing the RETURN key).

Conversely, the `value-changed` callback of a text gadget is invoked when the change to the gadget value is committed (again, usually by pressing the RETURN key).

The action required to “commit” a text change is defined by the back-end for the platform that you are writing for, and is not configurable.

Operations      `gadget-text`

Example      `contain(make(<text-field>, value-type: <integer>  
                  text: "1234"));`

See also      `<combo-box>`, page 481

`gadget-value-type`, page 525

`<password-field>`, page 550

`<text-editor>`, page 592

`<text-field>`, page 593

## `<tool-bar>`

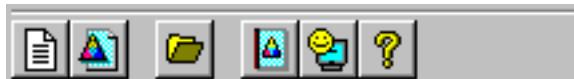
*Open abstract instantiable class*

Summary      The class of tool bars.

Superclasses    `<gadget> <multiple-child-composite-pane>`

Init-keywords    `update-callback:`  
                                An instance of type `<function>`.

Description      The class of tool bars. A tool bar is a gadget that contains, as children, a number of buttons that give the user quick access to the more common commands in an application. Typically, the label for each button is an icon that pictorially represents the operation that clicking the button performs.



**Figure 8.17** A tool bar

A tool bar is often placed underneath the menu bar of an application, although its position is very often configurable, and a tool bar may often be “docked” against any edge of the application’s frame. In addition, a tool bar can sometimes be displayed as a free-floating window inside the application.

Internally, this class maps into the Windows toolbar control.

Operations    `frame-tool-bar-setter make`

See also      `<button-box>`, page 473

&lt;status-bar&gt;, page 573

**<tree-control>***Open abstract instantiable class*

Summary      The class of tree controls.

Superclasses    &lt;collection-gadget&gt;

Init-keywords    **children-generator:**

An instance of type &lt;function&gt;.

**children-predicate:**

An instance of type &lt;function&gt;.

**icon-function:** An instance of type <function>.                  **show-edges?:** An instance of type <boolean>. Default  
                  value: #t.                  **show-root-edges?:** An instance of type <boolean>. Default  
                  value: #t.                  **show-buttons?:** An instance of type <boolean>. Default  
                  value: #t.                  **initial-depth:** An instance of type <integer>. Default  
                  value: 0.                  **scroll-bars:** An instance of type one-of(#f, #"none",  
                                  #"horizontal", #"vertical", #"both",  
                                  #"dynamic"). Default value: #"both".                  **popup-menu-callback:**

An instance of type &lt;function&gt;.

**key-press-callback:**                  An instance of type false-or(<command>,  
                                  <function>).

**roots:** An instance of type `<sequence>`. Default value: `#[]`.

Description The class of tree controls.



The **children-generator:** is the function that is used to generate the children below the root of the tree control. It is called with one argument, an object.

The **icon-function:** init-keyword lets you specify a function to supply icons for display in the control. The function is called with the item that needs an icon as its argument, and it should return an instance of `<image>` as its result. Typically, you might want to define an icon function that returns a different icon for each kind of item in the control. For example, if the control is used to display the files and directories on a hard disk, you would want to return the appropriate icon for each registered file type.

The **show-edges?::**, **show-root-edges?::**, and **show-buttons?::** init-keywords define whether lines are displayed for the edges of items in the tree control, the roots in the tree control, and whether the icons of items in the tree control are displayed, respectively. By default, all three are visible.

The number of levels of outline that are shown when the tree control is first displayed is controlled by the `initial-depth`: init-keyword. The default value of this is 0, meaning that only the top level of the outline is shown, with no nodes expanded.

The `scroll-bars`: init-keyword specifies whether the tree control has scroll bars or not.

You can use the `popup-menu-callback`: init-keyword to specify a context-sensitive menu to display for one or more selected items in the tree control. In Windows 95, for instance, such a context-sensitive menu can be displayed by right-clicking on any item or group of selected items in the list control.

The `key-press-callback`: init-keyword lets you specify a key-press callback. This type of callback is invoked whenever a key on the keyboard is pressed while the gadget has focus. For tree controls, a typical key-press callback might select an item in the control. See `gadget-key-press-callback`, page 502, for a fuller description of key-press callbacks.

The `roots`: init-keyword is used to specify any roots for the tree control. It is a sequence.

Internally, this class maps into the Windows tree view control.

Operations	<code>contract-node expand-node tree-control-children-predicate tree-control-children-predicate-setter tree-control-children-generator tree-control-children-generator-setter tree-control-roots tree-control-roots-setter</code>
------------	---

**Example**

```
make(<tree-control>,
      roots: #[1],
      children-generator:
        method (x) vector(x * 2, 1 + (x * 2)) end,
      icon-function: method (item :: <integer>)
        case
          odd?(item)  => $odd-icon;
          even?(item) => $even-icon;
        end);
```

**See also**

[add-node](#), page 468  
[find-node](#), page 485  
[make-node](#), page 540  
[remove-node](#), page 560

**tree-control-children-predicate***Generic function***Summary**

Returns the children predicate function of the specified tree control.

**Signature**

`tree-control-children-predicate tree-control  
=> children-predicate`

**Arguments**

*tree-control* An instance of type `<tree-control>`.

**Values**

*children-predicate* An instance of type `<function>`.

**Description**

Returns the children predicate function of *tree-control*.

**See also**

[<tree-control>](#), page 598  
[tree-control-children-predicate-setter](#), page 602  
[tree-control-children-generator](#), page 602

	<b>tree-control-children-predicate-setter</b>	<i>Generic function</i>
Summary	Sets the children predicate function of the specified tree control.	
Signature	<b>tree-control-children-predicate-setter</b> <i>children-predicate</i> <i>tree-control</i> => <i>children-predicate</i>	
Arguments	<i>children-predicate</i> An instance of type < <b>function</b> >. <i>tree-control</i> An instance of type < <b>tree-control</b> >.	
Values	<i>children-predicate</i> An instance of type < <b>function</b> >.	
Description	Sets the children predicate function of <i>tree-control</i> .	
See also	< <b>tree-control</b> >, page 598  <b>tree-control-children-predicate</b> , page 601  <b>tree-control-children-generator-setter</b> , page 603	

	<b>tree-control-children-generator</b>	<i>Generic function</i>
Summary	Returns the function that generates the children of the specified tree control.	
Signature	<b>tree-control-children-generator</b> <i>tree-control</i> => <i>children-generator</i>	
Arguments	<i>tree-control</i> An instance of type < <b>tree-control</b> >.	
Values	<i>children-generator</i> An instance of type < <b>function</b> >.	
Description	Returns the function that generates the children of <i>tree-control</i> . This is the function that is used to generate the children below the root of <i>tree-control</i> .	

See also      `<tree-control>`, page 598  
`tree-control-children-predicate`, page 601  
`tree-control-children-generator-setter`, page 603

## **tree-control-children-generator-setter**                          *Generic function*

Summary      Sets the function that generates the children of the specified tree control.

Signature     `tree-control-children-generator-setter`  
`children-generator tree-control`  
`=> children-generator`

Arguments     `children-generator`An instance of type `<function>`.  
`tree-control`      An instance of type `<tree-control>`.

Values        `children-generator`An instance of type `<function>`.

Description     Sets the function that generates the children of `tree-control`.  
This is the function that is used to generate the children below the root of `tree-control`.

See also      `<tree-control>`, page 598  
`tree-control-children-predicate-setter`, page 602  
`tree-control-children-generator`, page 602

## **tree-control-icon-function**                          *Generic function*

Summary      Returns the icon function for the specified list control.

Signature     `tree-control-icon-function tree-control => icon-function`

Arguments     `tree-control`      An instance of `<tree-control>`.

Values	<i>icon-function</i>	An instance of type < <b>function</b> >.
Description	<p>Returns the icon function for <i>tree-control</i>. This function lets you specify which icon to display for each item in the control. The function is called with the item that needs an icon as its argument, and it should return an instance of &lt;<b>image</b>&gt; as its result. Typically, you might want to define an icon function that returns a different icon for each kind of item in the control. For example, if the control is used to display the files and directories on a hard disk, you would want to return the appropriate icon for each registered file type.</p> <p>Note that, unlike list controls, the icon function for a tree control cannot be changed once the list control has been created.</p>	
See also	<p><a href="#">list-control-icon-function</a>, page 533</p> <p><a href="#">&lt;tree-control&gt;</a>, page 598</p>	

<b>tree-control-initial-depth</b>		<i>Generic function</i>
Summary	Returns the initial depth of the specified tree control.	
Signature	<b>tree-control-initial-depth</b> <i>tree-control</i> => <i>initial-depth</i>	
Arguments	<i>tree-control</i>	An instance of type < <b>tree-control</b> >.
Values	<i>initial-depth</i>	An instance of type < <b>integer</b> >.
Description	<p>Returns the initial depth of <i>tree-control</i>. This is the number of levels of outline that are visible in the tree control when it is first displayed. A return value of 0 indicates that only the top level of the outline is displayed initially. A return value of 1 indicates that outline is expanded to a depth of one (that is, any direct subnodes of the top level are displayed, but no others).</p>	

See also      **<tree-control>**, page 598  
                **tree-control-initial-depth-setter**, page 605

## **tree-control-initial-depth-setter** *Generic function*

Summary	Sets the initial depth of the specified tree control.	
Signature	<code>tree-control-initial-depth <i>initial-depth</i> <i>tree-control</i> =&gt; <i>initial-depth</i></code>	
Arguments	<i>initial-depth</i>	An instance of type <code>&lt;integer&gt;</code> .
	<i>tree-control</i>	An instance of type <code>&lt;tree-control&gt;</code> .
Values	<i>initial-depth</i>	An instance of type <code>&lt;integer&gt;</code> .
Description	Sets the initial depth of <i>tree-control</i> . This is the number of levels of outline that are visible in the tree control when it is first displayed. A return value of 0 indicates that only the top level of the outline is displayed initially. A return value of 1 indicates that outline is expanded to a depth of one (that is, any direct subnodes of the top level are displayed, but no others).	
See also	<code>&lt;tree-control&gt;</code> , page 598 <code>tree-control-initial-depth</code> , page 604	

## tree-control-roots *Generic function*

Summary	Returns the roots of the specified tree control.		
Signature	<code>tree-control-roots <i>tree</i> =&gt; <i>roots</i></code>		
Arguments	<table><tr><td><code><i>tree</i></code></td><td>An instance of type <code>&lt;tree-control&gt;</code>.</td></tr></table>	<code><i>tree</i></code>	An instance of type <code>&lt;tree-control&gt;</code> .
<code><i>tree</i></code>	An instance of type <code>&lt;tree-control&gt;</code> .		

Values      *roots*      An instance of type <sequence>.

Description    Returns the roots of *tree*.

Example     Create a tree control as follows:

```
*tree* := contain(make(<tree-control>,
                      roots: #(1, 2, 3),
                      children-generator:
                        method (x)
                          vector(x, x + 1)
                        end));
```

You can return the roots of this tree control as follows:

```
tree-control-roots(*tree*);
```

See also    <tree-control>, page 598

tree-control-roots-setter, page 606

## tree-control-roots-setter

*Generic function*

Summary     Sets the roots of the specified tree control.

Signature    `tree-control-roots-setter roots tree #key frame-manager => roots`

Arguments    *roots*      An instance of type <sequence>.

*tree*      An instance of type <tree-control>.

*frame-manager*    An instance of type <frame-manager>.

Values      *roots*      An instance of type <sequence>.

Description    Sets the roots of *tree*.

Example     Create a tree control without specifying any roots as follows:

```
*tree* := contain(make(<tree-control>,
                      children-generator:
                        method (x)
                          vector(x, x + 1)
                        end));
```

You can set the roots of this tree control as follows:

```
tree-control-roots(*tree*) := #(1, 2, 3);
```

The tree control is updated on the screen to reflect this change.

See also      [<tree-control>](#), page 598

[tree-control-roots](#), page 605

## <tree-node> *Open abstract instantiable class*

Summary      The class of nodes in tree controls.

Superclasses      [<object>](#)

Init-keywords      **parent-nodes:** An instance of type [<sequence>](#).

**child-nodes:** An instance of type [<sequence>](#).

**generation:** An instance of type [<integer>](#). Default value: 0.

**object:** An instance of type [<object>](#).

Description      The class of nodes in tree controls. A tree node represents an object, and is displayed as a text label accompanied by an icon. Tree nodes are analogous to list items in a list control or table items in a table control.

To the left of a tree node is a small plus or minus sign. If a plus sign is displayed, this indicates that the node contains subnodes that are currently not visible. If a minus sign is displayed, this indicates either that the node does not contain any subnodes, or that the subnodes are already visible.

The `parent-nodes:` and `child-nodes:` init-keywords let you specify any parents and children that the node has.

The `object:` init-keyword specifies the object that is represented by the tree node. For example, in the case of a file manager application, this might be a directory on disk.

Operations	<code>contract-node expand-node node-children node-expanded? node-parents</code>
See also	<code>&lt;tree-control&gt;</code> , page 598

## update-gadget

*Generic function*

Summary	Forces the specified gadget to be redrawn.
Signature	<code>update-gadget gadget =&gt; ()</code>
Arguments	<code>gadget</code> An instance of type <code>&lt;gadget&gt;</code> .
Values	None
Description	Forces <code>gadget</code> to be redrawn. This can be useful if a number of changes have been made which have not been reflected in the gadget automatically (for example, by using pixmaps to perform image operations)

## `<value-gadget>`

*Open abstract class*

Summary	The class of gadgets that can have values.
---------	--

Superclasses	<code>&lt;gadget&gt;</code>
Init-keywords	<p><code>value:</code> An instance of type <code>&lt;object&gt;</code>.</p> <p><code>value-changed-callback:</code> An instance of type <code>false-or(&lt;command&gt;, &lt;function&gt;)</code>.</p>
Description	<p>The class of gadgets that can take values.</p> <p>The <code>value:</code> init-keyword specifies the current gadget value. For tab controls, if the gadget ID is specified, then that is passed as the gadget value whether or not <code>value:</code> is specified.</p> <p>The <code>value-changed-callback:</code> init-keyword is the callback that is invoked when the gadget value has changed, such as when a scroll bar slug has come to rest after being dragged, or when the changes to text in a text field have been committed by pressing the RETURN key.</p>
Operations	<code>gadget-value</code> <code>gadget-value-changed-callback</code> <code>gadget-value-changed-callback-setter</code> <code>gadget-value-setter</code> <code>gadget-value-type</code>
Example	
See also	<p><code>gadget-value</code>, page 517</p> <p><code>gadget-value-changed-callback</code>, page 519</p>

**<value-range-gadget>***Open abstract class*

Summary	The class of all value gadgets with ranges.
Superclasses	<code>&lt;value-gadget&gt;</code>

Init-keywords	<b>value-range:</b> An instance of type <code>&lt;range&gt;</code> . Default value: <code>range(from: 0, to: 100)</code> .
Description	The class of all value gadgets with ranges. You should not create a direct instance of this class.  The <b>value-range:</b> init-keyword is the range of values that the gadget value of a value range gadget can take. This may be different in any given situation: when downloading a file or compiling source code, you might want to use a value range of 0-100, to indicate percentage done (this is the default). When downloading e-mail messages from a mail server, however, you may want to use a range equal to the number of messages being downloaded.
Operations	<code>gadget-value-range</code> <code>gadget-value-range-setter</code>
Example	<code>contain(make(&lt;slider&gt;,                   value-range:                   range(from: -20, to: 20, by: 5)));</code>
See also	<code>&lt;progress-bar&gt;</code> , page 551 <code>&lt;scroll-bar&gt;</code> , page 561 <code>&lt;slider&gt;</code> , page 567 <code>&lt;value-gadget&gt;</code> , page 608

<b>&lt;viewport&gt;</b>		<i>Open abstract instantiable class</i>
Summary	The class of viewports.	
Superclasses	<code>&lt;gadget&gt;</code> <code>&lt;single-child-composite-pane&gt;</code>	
Init-keywords	<code>horizontal-scroll-bar:</code>  An instance of type <code>false-or(&lt;scroll-bar&gt;)</code> . Default value: <code>#f</code> .	

**`vertical-scroll-bar:`**

An instance of type `false-or(<scroll-bar>)`. Default value: `#f`.

**Description**

The class of viewports. A viewport is a sheet “through” which other sheets are visible; they are used to implement a clipping region for scrolling.

The `horizontal-scroll-bar:` and `vertical-scroll-bar:` init-keywords specify whether the viewport has horizontal and vertical scroll bars, respectively.

In most applications, you should not need to use a viewport yourself. However, there are some circumstances in which defining your own viewports is invaluable. In particular, if you need to use a single scroll bar to scroll more than one window at the same time, you should define each window as a viewport, and use the same scroll bar to scroll each window. There are two situations where this behavior is quite common:

- In applications which have vertical or horizontal rulers around a document window, such as a drawing application. In this case, the rulers must scroll with the drawing itself.
- In applications such as spreadsheets, where row and column headings need to scroll with the document. Note that you may also choose to implement this kind of functionality using a table control.

**Operations**

`viewport-region`

**See also**

`sheet-viewport`, page 566

`sheet-viewport-region`, page 567

`viewport?`, page 612

`viewport-region`, page 612

## viewport?

*Generic function*

**Summary** Returns true if the specified object is a viewport.

**Signature** `viewport? object => viewport?`

**Arguments** `object` An instance of type `<object>`.

**Values** `viewport?` An instance of type `<boolean>`.

**Description** Returns true if `object` is a viewport.

**Example** To test whether the gadget `*gadget*` is a viewport:

```
viewport?(*gadget*);
```

**See also** `<viewport>`, page 610

`<button-box>`, page 473

`<border>`, page 469

## viewport-region

*Generic function*

**Summary** Returns the region for the specified viewport.

**Signature** `viewport-region viewport => region`

**Arguments** `viewport` An instance of type `<viewport>`.

**Values** `region` An instance of type `<region>`.

**Description** Returns the region for `viewport`.

**Example** To return the region for a viewport `*viewer*`:

```
viewport-region(*viewer*);
```

**See also** `<viewport>`, page 610

**with-border***Statement macro*

Summary	Creates the specified sheet and places a border around it.				
Macro call	<code>with-border ([options]) {pane} end</code>				
Arguments	<table> <tr> <td><i>options</i></td><td>Dylan arguments<sub>bnf</sub></td></tr> <tr> <td><i>pane</i></td><td>A Dylan expression<sub>bnf</sub></td></tr> </table>	<i>options</i>	Dylan arguments <sub>bnf</sub>	<i>pane</i>	A Dylan expression <sub>bnf</sub>
<i>options</i>	Dylan arguments <sub>bnf</sub>				
<i>pane</i>	A Dylan expression <sub>bnf</sub>				
Values	None.				
Description	<p>Creates <i>pane</i> with a border around it, taking into account any of the specified <i>options</i>.</p> <p>The options specified may be any of the legal init-keywords used to specify an instance of &lt;border&gt;. If no options are specified, then the default border is used.</p> <p>The pane is an expression whose return value is the sheet around which a border should be placed.</p>				
Example	To create a button in a border:				
	<pre>contain(with-border (type: #"raised")            make(&lt;button&gt;,                  label: "Hello") end);</pre>				
See also	<p>&lt;border&gt;, page 469</p> <p>labelling, page 529</p> <p>with-spacing, page 613</p>				

**with-spacing***Statement macro*

Summary	Creates the specified sheet and places spacing around it.
Macro call	<code>with-spacing ([options]) {pane} end</code>

Arguments	<i>options</i>	Dylan arguments <sub>bnf</sub> .
	<i>pane</i>	A Dylan expression <sub>bnf</sub> .
Values	None.	
Description		<p>Creates <i>pane</i> with spacing around it, taking into account any of the specified <i>options</i>.</p> <p>The options specified may be any of the legal init-keywords used to specify an instance of &lt;<code>spacing</code>&gt;. If no options are specified, then the default spacing is used.</p> <p>The pane is an expression whose return value is the sheet around which spacing should be placed.</p>
Example		<pre>contain(with-spacing (thickness: 10)            (vertically () make(&lt;button&gt;,                                label: "Hello");             make(&lt;button&gt;,                   label: "World"))            end) end);</pre>
See also		<p>&lt;<code>null-pane</code>&gt;, page 420</p> <p>&lt;<code>spacing</code>&gt;, page 569</p> <p><code>with-border</code>, page 613</p>

# 9

---

---

# DUIM-Frames Library

## 9.1 Overview

The DUIM-Frames library contains interfaces that define a wide variety of frames for use in your GUI applications, as well as the necessary functions, generic functions, and macros for creating and manipulating them. The library contains a single module, `duim-frames`, from which all the interfaces described in this chapter are exposed. Section 9.4 on page 621 contains complete reference entries for each exposed interface.

Frames are the basic components used to display DUIM objects on-screen. An instance of type `<frame>` is an object representing some state in a user application, plus the sheets in its interface. Frames control the overall appearance of the entire window, allowing you to distinguish, for example, between a normal window and a dialog box, or allowing you to specify modal or modeless dialog boxes, and might include such things as a menu bar, a tool bar, and a status bar.

Frames exist on windows and contain sheets, which can be instances of `<layout>` or `<gadget>`, or any of their subclasses, and an event loop. The event loop associated with a frame is represented by an instance of a subclass of `<event>`. An overview of these subclasses is provided in Section 9.2.3 on page 619.

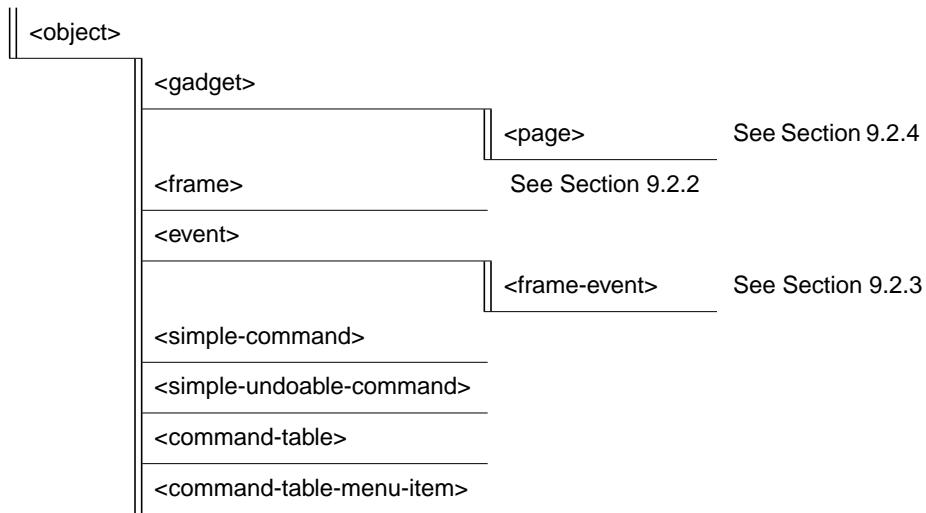
## 9.2 The class hierarchy for DUIM-Frames

This section presents an overview of the available classes of frame, frame event, and command-related classes, and describes the class hierarchy present.

### 9.2.1 The `<frame>` class and its subclasses

The base class for all DUIM frames is the `<frame>` class, which is itself a subclass of `<object>`. In addition, there are a number of classes related to commands that are subclasses of `<object>`, together with a number of classes related to events that occur in frames. Table 9.1 shows the overall class hierarchy for the base classes exported by the DUIM-Frames library.

**Table 9.1** Overall class hierarchy for the DUIM-Frames library



The `<frame>` class represents the base class for all types of frame. An introduction to the subclasses available is given in Section 9.2.2.

The `<event>` class represents the base class for all events that can occur. Although this class and the `<frame-event>` subclass are exposed by the DUIM-Sheets library, the subclasses of `<frame-event>` itself are exposed by

the DUIM-Frames library. See Section 9.2.3 on page 619 for an introduction to these subclasses. See Chapter 5, “DUIM-Sheets Library”, for a complete description of the DUIM-Sheets library.

The remaining four classes exposed by the DUIM-Frames library relate to commands and their use in application menus.

**<simple-command>**

This class is used to create the most basic type of command. A command is an operation that can be invoked as a callback from a menu item, a button, or other suitable interface control.

**<simple-undoable-command>**

This class is used to define commands whose effects can be reversed. Typically, the user chooses the command **Edit > Undo** to reverse the effects of a command of this class.

**<command-table>**

The **<command-table>** class is used to define the complete menu structure of an application frame, from the menu bar and menus to the menu items on each menu.

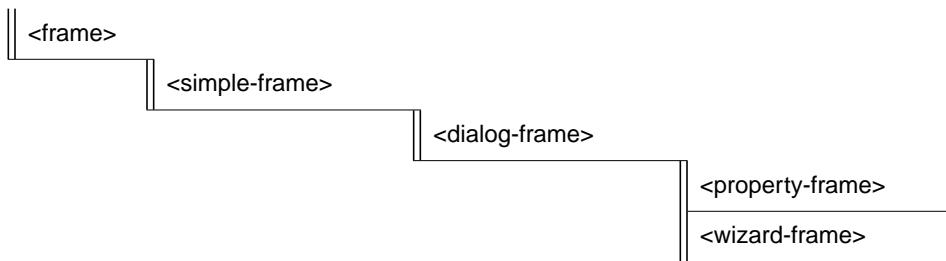
**<command-table-menu-item>**

This class represents a menu item on a menu defined in a command table.

### 9.2.2 Subclasses of <frame>

A number of subclasses of <frame> are provided to allow you to create a variety of common types of frame. These subclasses are shown in Table 9.2.

**Table 9.2** Subclasses of the <frame> class



**<simple-frame>** This class is the most common sort of frame and is used to create a standard window in an application.

**<dialog-frame>** This class is used to create dialog boxes for use in an application.

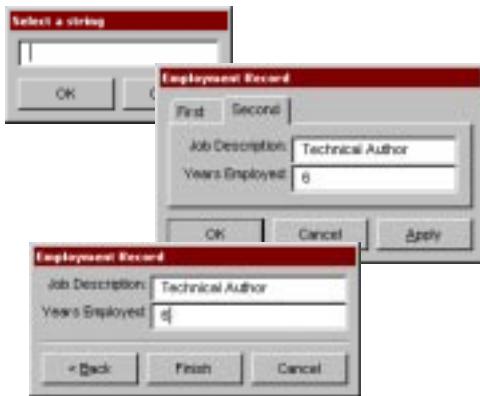
**<property-frame>**

This class is used to create property sheets for use in an application. Property sheets are a special type of dialog box which make use of tab controls to display several pages of information within the same dialog.

**<wizard-frame>**

This class is used to create wizards for use in an application. Wizards are a special type of multi-page dialog in which the user is guided through a series of sequen-

tial steps, filling out any information requested and using **Next** and **Back** buttons to navigate to the next or previous steps in the process.

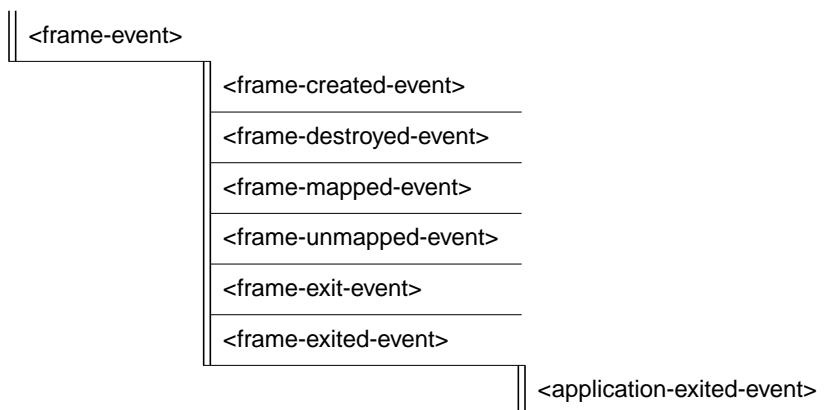


**Figure 9.1** A dialog frame, a property frame, and a wizard frame

### 9.2.3 Subclasses of <frame-event>

The `<frame-event>` class provides a number of subclasses that describe various events that can occur in frames. These subclasses are shown in Table 9.3.

**Table 9.3** Subclasses of the `<frame-event>` class



The name of each of these subclasses accurately reflects the type of event that they are used to represent. The classes `<frame-created-event>` and `<frame-destroyed-event>` represent a frame being created or destroyed. The classes `<frame-mapped-event>` and `<frame-unmapped-event>` represent the events that occur when a frame is displayed on the computer screen or removed from it. The class `<frame-exit-event>` represents the act of exiting a frame, and the class `<frame-exited-event>` represents the event where a frame has been successfully exited.

In addition, the class `<frame-exited-event>` has a subclass `<application-exited-event>`. This is reserved for the special case where the frame that has been exited is actually the parent frame for the whole application, in which case the whole application is exited, together with any other frames that may have been spawned as a result of using the application.

**Note:** The classes `<frame-mapped-event>` and `<frame-unmapped-event>` are distinct from the classes `<frame-created-event>` and `<frame-destroyed-event>`. A frame is not necessarily mapped as soon as it is created, and any frame can be unmapped from the screen without actually destroying it (for example, a frame may be iconized).

#### **9.2.4 Subclasses of `<page>`**

Although the `<page>` class is itself a subclass of `<gadget>`, and is exposed by the DUIM-Gadgets library, two of its subclasses are exposed by the DUIM-Frames library: `<wizard-page>` and `<property-page>`. See Section 8.4.3 on page 454 for an introduction to these classes.

### **9.3 DUIM-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 this chapter.

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, 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,
```

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

## 9.4 DUIM-Frames Module

This section contains a complete reference of all the interfaces that are exported from the `duim-frames` module.

=

*G.f. method*

Summary      Returns true if the specified commands are the same.

Signature      = `command1 command2 => equal?`

Arguments      `command1`      An instance of type `<command>`.

`command2`      An instance of type `<command>`.

Values      `equal?`      An instance of type `<boolean>`.

Description      Returns true if `command1` and `command2` are the same.

**add-command**

*Generic function*

Summary      Adds a command to the specified command table.

Signature	<code>add-command command-table <i>command</i> #key <i>name menu image accelerator mnemonic error?</i> =&gt; ()</code>
Arguments	<p><i>command-table</i> An instance of type <code>&lt;command-table&gt;</code>.</p> <p><i>command</i> An instance of type <code>type-union(&lt;command&gt;, &lt;function&gt;)</code>.</p> <p><i>name</i> An instance of type <code>false-or(&lt;string&gt;)</code>.</p> <p><i>menu</i> An instance of type <code>false-or(&lt;menu&gt;)</code>.</p> <p><i>image</i> An instance of type <code>false-or(&lt;image&gt;)</code>.</p> <p><i>accelerator</i> An instance of type <code>false-or(&lt;gesture&gt;)</code>.</p> <p><i>mnemonic</i> An instance of type <code>false-or(&lt;gesture&gt;)</code>.</p> <p><i>error?</i> An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</p>
Values	None
Description	<p>You can supply a keyboard accelerator or a mnemonic using the <i>accelerator</i> and <i>mnemonic</i> arguments respectively.</p> <p>Adds <i>command</i> to <i>command-table</i>.</p> <p>The argument <i>name</i> is the command-line name for the command.</p> <ul style="list-style-type: none"> <li>• When <i>name</i> is <code>#f</code>, the command is not available via command-line interactions.</li> <li>• When <i>name</i> is a string, that string is the command-line name for the command.</li> </ul> <p>For the purposes of command-line name lookup, the character case of <i>name</i> is ignored.</p> <p>The argument <i>menu</i> is a menu for <i>command</i>.</p> <ul style="list-style-type: none"> <li>• When <i>menu</i> is <code>#f</code>, <i>command</i> is not available via menus.</li> <li>• When <i>menu</i> is a string, the string is used as the menu name.</li> </ul>

- When *menu* is `#t` and *name* is a string, then *name* is used as the menu name.
- When *menu* is `#t` and *name* is not a string, a menu name is automatically generated.
- When *menu* is a list of the form `(string, menu-options)`, *string* is the menu name and *menu-options* consists of a list of keyword-value pairs. Each keyword-value pair is itself a list. The valid keywords are `after:`, `documentation:`, and `text-style:`, which are interpreted as for `add-command-table-menu-item`.

You can supply an image that will appear on the menu next to the command name using the *image* argument. When supplying an image, bear in mind the size of the menu: you should only supply a small icon-sized image for a menu command. There may also be other interface guidelines that you wish to follow when using images in menu items.

The value for *accelerator* is either keyboard gesture or `#f`. When it is a gesture, this gesture represents the keystroke accelerator for the command; otherwise the command is not available via keystroke accelerators. Similarly, if *mnemonic* is supplied, this gesture is used as a mnemonic for the command.

If *command* is already present in the command table and *error?* is `#t`, an error is signalled. When *command* is already present in the command table and *error?* is `#f`, then the old command-line name, menu, and keystroke accelerator are removed from the command table before creating the new one.

See also [remove-command](#), page 743

## `add-command-table-menu-item`

*Generic function*

Summary Adds a menu item to the specified command table.

Signature	<code>add-command-table-menu-item command-table string type value #key documentation after accelerator mnemonic text-style error? items label-key value-key test callback =&gt; menu-item</code>
Arguments	<p><i>command-table</i> An instance of type <code>&lt;command-table&gt;</code>.</p> <p><i>string</i> An instance of type <code>false-or(&lt;string&gt;)</code>.</p> <p><i>type</i> An instance of type <code>one-of(#"command", #"function", #"menu", #"divider")</code>.</p> <p><i>value</i> An instance of type <code>&lt;object&gt;</code>.</p> <p><i>documentation</i> An instance of type <code>&lt;string&gt;</code>.</p> <p><i>after</i> An instance of type <code>one-of(#"start", #"end", #"sort")</code>, or an instance of <code>&lt;string&gt;</code>. Default value: <code>#"end"</code>.</p> <p><i>accelerator</i> An instance of type <code>false-or(&lt;gesture&gt;)</code>.</p> <p><i>mnemonic</i> An instance of type <code>false-or(&lt;gesture&gt;)</code>.</p> <p><i>text-style</i> An instance of type <code>&lt;text-style&gt;</code>.</p> <p><i>error?</i> An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</p> <p><i>items</i> An instance of type <code>limited(&lt;sequence&gt;, of: )</code>.</p> <p><i>label-key</i> An instance of type <code>&lt;function&gt;</code>.</p> <p><i>value-key</i> An instance of type <code>&lt;function&gt;</code>.</p> <p><i>test</i> An instance of type <code>&lt;function&gt;</code>.</p> <p><i>callback</i> An instance of type <code>&lt;function&gt;</code>.</p>
Values	<i>menu-item</i> An instance of type <code>&lt;command-table-menu-item&gt;</code> .
Description	Adds a command menu item to the menu in <i>command-table</i> . The <i>string</i> argument is the name of the command menu item; its character case is ignored. The <i>type</i> of the item is either <code>#"command", #"function", #"menu",</code> or <code>#"divider"</code> .

When *type* is `#"command"`, *value* must be one of the following:

- A command (a list consisting of a command name followed by a list of the arguments for the command).
- A command name. In this case, *value* behaves as though a command with no arguments was supplied.

When all the required arguments for the command are supplied, clicking on an item in the menu invokes the command immediately. Otherwise, the user is prompted for the remaining required arguments.

When *type* is `#"function"`, *value* must be a function having indefinite extent that, when called, returns a command. The function is called with two arguments:

- The gesture used to select the item (either a keyboard or button press event).
- A “numeric argument”.

When *type* is `#"menu"`, this indicates that a sub-menu is required, and *value* must be another command table or the name of another command table.

When *type* is `#"divider"`, some sort of a dividing line is displayed in the menu at that point. If *string* is supplied, it will be drawn as the divider instead of a line. If the look and feel provided by the underlying window system has no corresponding concept, `#"divider"` items may be ignored. When *type* is `#"divider"`, *value* is ignored.

The argument *documentation* specifies a documentation string. This can be used to provide the user with some online documentation for the menu item. Documentation strings are often displayed in a status bar at the bottom of an application; highlighting the menu item using the mouse pointer displays the documentation string in the status bar.

The *text-style* argument, if supplied, represents text style. This specifies the font family, style, and weight with which to display the menu item in the menu. For most menu items,

you should just use the default text style (that is, the one that the user chooses for all applications). However, in certain cases, some variation is allowed.

The *text-style* argument is of most use in context sensitive pop-up menus, which often have a default menu item. This is usually the command that is invoked by pressing the RETURN key on the current selection: for example, in a list of files, the default command usually opens the selected file in the application associated with it. In Windows 95, the default command is displayed using a bold font, to differentiate it from other commands in the menu, and you should use the *text-style* argument to specify this.

When altering the text style of a menu item, you should always try to stick to any relevant interface guidelines.

The *items* argument is used to specify the gadgets that are to be supplied to the command table as menu items. You can supply either push boxes, check boxes, or radio boxes.

The *after* argument denotes where in the menu the new item is to be added. It must be one of the following:

**#"start"** Adds the new item to the beginning of the menu.

**#"end"** Adds the new item to the end of the menu.

A string naming an existing entry

Adds the new item after that entry.

**#"sort"** Insert the item in such a way as to maintain the menu in alphabetical order.

If *mnemonic* is supplied, the item is added to the keyboard mnemonic table for the command table. The value of *mnemonic* must be a keyboard gesture name.

When *mnemonic* is supplied and *type* is **"command"** or **"function"**, typing a key on the keyboard that matches the mnemonic invokes the command specified by *value*.

When *type* is `#"menu"`, the command is read from the submenu indicated by *value* in a window system specific manner. This usually means that the submenu itself is displayed, allowing the user to see the available options at that point.

When *accelerator* is supplied, typing a key sequence on the keyboard that matches the accelerator invokes the command specified by *value*, no matter what *type* is.

If the item named by *string* is already present in the command table and *error?* is `#t`, then an error is signalled. When the item is already present in the command table and *error?* is `#f`, the old item is removed from the menu before adding the new item. Note that the character case of *string* is ignored when searching the command table.

See also [`<command-table-menu-item>`](#), page 637

[`remove-command-table-menu-item`](#), page 744

## **<application-exited-event>**

*Sealed instantiable class*

Summary The class of events signalled when an application exits.

Superclasses [`<frame-exited-event>`](#)

Init-keywords None.

Description The class of events signalled when an application exits. An instance of this class is distributed when your application is exited, for instance by choosing **File > Exit** from its main menu bar.

Operations None.

See also [`exit-frame`](#), page 692

[`<frame-exited-event>`](#), page 704

**apply-in-frame** *Generic function*

Summary	Applies the specified function to the given arguments in the main thread of the frame.	
Signature	<code>apply-in-frame frame function arg #rest args =&gt; ()</code>	
Arguments	<i>frame</i>	An instance of type < <code>frame</code> >.
	<i>function</i>	An instance of type < <code>function</code> >.
	<i>arg</i>	An instance of type < <code>object</code> >.
	<i>args</i>	Instances of type < <code>object</code> >.
Values	None.	
Description	Applies <i>function</i> to the given arguments in the main thread of <i>frame</i> . You must supply at least one argument ( <i>arg</i> ), though you can optionally supply as many additional arguments as you like.	
See also	<a href="#">call-in-frame</a> , page 628	

**call-in-frame** *Generic function*

Summary	Calls the specified function with the given arguments in the main thread of the frame.	
Signature	<code>call-in-frame frame function #rest args =&gt; ()</code>	
Arguments	<i>frame</i>	An instance of type < <code>frame</code> >.
	<i>function</i>	An instance of type < <code>function</code> >.
	<i>args</i>	Instances of type < <code>object</code> >.
Values	None.	

Description Calls *function* with the given arguments in the main thread of *frame*.

See also [apply-in-frame](#), page 628

## cancel-dialog

### *Generic function*

Summary Cancels the specified dialog.

Signature `cancel-dialog dialog #key destroy? => ()`

Arguments *dialog* An instance of type `<dialog-frame>`.

*destroy?* An instance of type `<boolean>`. Default value: `#t`.

Values None

Description Cancels *dialog* and removes it from the screen. Any changes that the user has made to information displayed in the dialog is discarded.

If *destroy?* is `#t` then the dialog is unmapped from the screen. This is the default callback used for the cancel button in a dialog.

Example The following example defines a button, `*no-button*`, that calls `cancel-dialog` as its activate-callback. This button is then used in a dialog that simply replaces the standard cancel button for the newly defined dialog. Note that the example assumes the existence of a similar `*yes-button*` to replace the exit button.

```
define variable *no-button*
  = make(<push-button>, label: "No",
         activate-callback: cancel-dialog,
         max-width: $fill);
```

```
make(<dialog-frame>,
      exit-button?: #f,
      cancel-button?: #f,
      layout: vertically ()
          make(<label>,
              label: "Simple dialog");
      horizontally ()
          *yes-button*;
          *no-button*;
      end
  end);

start-frame(*dialog*);
```

See also      [dialog-cancel-callback](#), page 662  
[<dialog-frame>](#), page 670  
[start-dialog](#), page 748  
[exit-dialog](#), page 691

## **clear-progress-note**

*Generic function*

Summary      Clears the specified progress note.

Signature      `clear-progress-note framem progress-note => ()`

Arguments      `framem`      An instance of type [<frame-manager>](#).  
`progress-note`      An instance of type [<progress-note>](#).

Values      None

Description      Clears the specified progress note.

## **<command>**

*Open abstract instantiable class*

Summary      The class of commands.

Superclasses	<code>&lt;object&gt;</code>
Init-keywords	<p><b>function:</b> An instance of type <code>&lt;function&gt;</code>.</p> <p><b>arguments:</b> An instance of type <code>&lt;sequence&gt;</code>. Default value: <code>#[]</code>.</p>
Description	<p>The class of commands. These are commands that can be grouped together in a command table to form the set of commands available to an application (available, for example, from the menu bar of the application). The resulting command object can then be executed by calling <code>execute-command</code>.</p> <p>The <b>function:</b> init-keyword is the command function that is called by the command object. A command function is rather like a callback to a <code>&lt;command&gt;</code> object: a command can be executed via <code>execute-command</code>, which then invokes the command function. Command functions take at least one argument: a <code>&lt;frame&gt;</code> object.</p> <p>The <b>arguments:</b> init-keyword are the arguments passed to the command function.</p>
Operations	<code>= add-command command-arguments command-enabled?</code> <code>command-enabled?-setter command-function command-</code> <code>undoable? dialog-cancel-callback-setter dialog-exit-</code> <code>callback-setter execute-command gadget-command</code> <code>gadget-command-setter gadget-key-press-callback-</code> <code>setter redo-command remove-command undo-command</code>
See also	<a href="#">command?</a> , page 632 <a href="#">command-arguments</a> , page 632 <a href="#">command-function</a> , page 634 <a href="#">execute-command</a> , page 690 <a href="#">&lt;simple-command&gt;</a> , page 745

**command?** *Generic function*

Summary	Returns true if the specified object is a command.	
Signature	<code>command? object =&gt; command?</code>	
Arguments	<i>object</i>	An instance of type <object>.
Values	<i>command?</i>	An instance of type <boolean>.
Description	Returns true if <i>object</i> is an instance of <command>.	
See also	<command>, page 630	

**command-arguments** *Generic function*

Summary	Returns the arguments to the specified command.	
Signature	<code>command-arguments command =&gt; arguments</code>	
Arguments	<i>command</i>	An instance of type <command>.
Values	<i>arguments</i>	An instance of type <sequence>.
Description	Returns the arguments to <i>command</i> .	
See also	<command>, page 630	

**command-enabled?** *Generic function*

Summary	Returns true if the specified command is enabled.	
Signature	<code>command-enabled? command frame #key =&gt; enabled?</code>	

Arguments	<i>command</i>	An instance of type <code>type-union(&lt;command&gt;, &lt;command-table&gt;)</code> .
	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> .
Values	<i>enabled?</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description		Returns true if <i>command</i> in <i>frame</i> is enabled.
See also		<code>&lt;command&gt;</code> , page 630 <code>command-enabled?-setter</code> , page 633

## command-enabled?-setter *Generic function*

Summary	Enables or disables the specified command.	
Signature	<code>command-enabled?-setter enabled? command frame =&gt; enabled?</code>	
Arguments	<i>enabled?</i>	An instance of type <code>&lt;boolean&gt;</code> .
	<i>command</i>	An instance of type <code>type-union(&lt;command&gt;, &lt;command-table&gt;)</code> .
	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> .
Values	<i>enabled?</i>	An instance of type <code>&lt;boolean&gt;</code> .
Description	<p>Enables or disables <i>command</i> in <i>frame</i>. If <i>enabled?</i> is true, then <i>command</i> is enabled, otherwise it is disabled. Enabling and disabling a command enables and disables all the gadgets that are associated with the command, such as menu items and tool bar buttons.</p> <p>This function is useful when manipulating the disabled commands in <i>frame</i>. For example, it is common to disable the <b>Save</b> menu command immediately after saving a file, enabling it again only when the file has been modified.</p>	

See also [command-enabled?](#), page 632

command-function		Generic function
Summary	Returns the function associated with the specified command.	
Signature	<code>command-function command =&gt; function</code>	
Arguments	<code>command</code>	An instance of type <code>&lt;command&gt;</code> .
Values	<code>function</code>	An instance of type <code>&lt;function&gt;</code> .
Description	Returns the function associated with <i>command</i> . A command function is the function that is called by a <code>&lt;command&gt;</code> object. Command functions are similar to callbacks, in that they are user functions that are invoked in order to perform some action. Command functions take at least one argument: a <code>&lt;frame&gt;</code> object.	
See also	<code>&lt;command&gt;</code> , page 630 <code>execute-command</code> , page 690	

<command-table>		<i>Open abstract instantiable class</i>
Summary	The class of command tables.	
Superclasses	<object>	
Init-keywords	<b>name:</b>	An instance of type <object>. Required.
	<b>inherit-from:</b>	An instance of type limited(<sequence>, of: <command-table>). Required.
	<b>resource-id:</b>	An instance of type false-or(<object>). Default value: #f.

Description	<p>The class of command tables. The command table for an application gives a complete specification of the commands available to that application, through its menus, tool bars, mnemonics, and accelerators.</p> <p>The <code>name:</code> init-keyword is a symbol that names the current command table.</p> <p>The <code>inherit-from:</code> init-keyword is a sequence of command tables whose behavior the current command table should inherit. All command tables inherit the behavior of the command table specified by <code>*global-command-table*</code>, and can also inherit the behavior specified by <code>*user-command-table*</code>.</p> <p>You do not normally need to specify a unique <code>resource-id:</code> yourself. As with most other DUIM classes, the <code>name:</code> init-keyword serves as a sufficient unique identifier.</p>
Operations	<pre>add-command add-command-table-menu-item command- table-accelerators command-table-commands command- table-menu command-table-name frame-command-table- setter make make-menu-from-command-table-menu make- menus-from-command-table remove-command remove-command-table remove-command-table-menu-item</pre>
Example	<pre>define command-table *clipboard-command-table*   =(*global-command-table*)     menu-item "Cut"      = cut-selection,       documentation: \$cut-doc;     menu-item "Copy"     = copy-selection,       documentation: \$copy-doc;     menu-item "Paste"    = paste-from-clipboard,       documentation: \$paste-doc;     menu-item "Delete"   = delete-selection,       documentation: \$delete-doc; end command-table *clipboard-command-table*;</pre>
See also	<p><code>*global-command-table*</code>, page 725</p> <p><code>*user-command-table*</code>, page 750</p>

**command-table?** *Generic function*

Summary	Returns true if the specified object is a command table.
Signature	<code>command-table? object =&gt; command-table?</code>
Arguments	<code>object</code> An instance of type <code>&lt;object&gt;</code> .
Values	<code>command-table?</code> An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if <i>object</i> is a command table.
See also	<code>&lt;command-table&gt;</code> , page 634

**command-table-accelerators** *Generic function*

Summary	Returns the keyboard accelerators for the specified command table.
Signature	<code>command-table-accelerators command-table =&gt; accelerators</code>
Arguments	<code>command-table</code> An instance of type <code>&lt;command-table&gt;</code> .
Values	<code>accelerators</code> An instance of type <code>limited(&lt;sequence&gt;, of: &lt;gesture&gt;)</code> .
Description	Returns the keyboard accelerators for <i>command-table</i> .
See also	<code>command-table-commands</code> , page 636

**command-table-commands** *Generic function*

Summary	Returns the commands for the specified command table.
Signature	<code>command-table-commands command-table =&gt; commands</code>

Arguments	<i>command-table</i>	An instance of type <command-table>.
Values	<i>commands</i>	An instance of type limited(<sequence>, of: <command>).
Description		Returns the commands defined for <i>command-table</i> .
See also		<code>command-table-accelerators</code> , page 636 <code>command-table-menu</code> , page 637

**command-table-menu***Generic function*

Summary		Returns the menu items in the specified command table.
Signature		<code>command-table-menu command-table =&gt; menu-items</code>
Arguments	<i>command-table</i>	An instance of type <command-table>.
Values	<i>menu-items</i>	An instance of type <stretchy-vector>.
Description		Returns the menu items in <i>command-table</i> .
See also		<code>command-table-commands</code> , page 636 <code>command-table-name</code> , page 639

**<command-table-menu-item>***Sealed instantiable class*

Summary		The class of menu items in command tables.
Superclasses		<object>
Init-keywords	<code>name:</code>	An instance of type <code>false-or(&lt;string&gt;)</code> . Default value: #f.

<b>image:</b>	An instance of type <code>false-or(type-union(&lt;string&gt;, &lt;image&gt;))</code> . Default value: <code>#f</code>
<b>type:</b>	An instance of type <code>one-of(#"command", "#function", "#menu", "#divider")</code> .
<b>value:</b>	An instance of type <code>&lt;object&gt;</code> . Default value: <code>#f</code> .
<b>options:</b>	An instance of type <code>&lt;sequence&gt;</code> . Default value: <code>#()</code> .
<b>accelerator:</b>	An instance of type <code>false-or(&lt;gesture&gt;)</code> . Default value: <code>#f</code> .
<b>mnemonic:</b>	An instance of type <code>false-or(&lt;gesture&gt;)</code> . Default value: <code>#f</code> .
<b>Description</b>	<p>The class of menu items in command tables. This class models menu items, tool bar items, accelerators, and mnemonics for a command table entry.</p> <p>The <code>type:</code> init-keyword denotes what type of menu item has been created. This is either <code>#"command"</code>, <code>#"function"</code>, <code>#"menu"</code>, or <code>#"divider"</code>.</p> <p>When <code>type:</code> is <code>#"command"</code>, <code>value:</code> must be one of the following:</p> <ul style="list-style-type: none"> <li>• A command (a list consisting of a command name followed by a list of the arguments for the command).</li> <li>• A command name. In this case, <code>value</code> behaves as though a command with no arguments was supplied.</li> </ul> <p>When all the required arguments for the command are supplied, clicking on an item in the menu invokes the command immediately. Otherwise, the user is prompted for the remaining required arguments.</p>

When `type:` is `#"function"`, `value:` must be a function having indefinite extent that, when called, returns a command. The function is called with two arguments:

- The gesture used to select the item (either a keyboard or button press event).
- A “numeric argument”.

When `type:` is `#"menu"`, this indicates that a sub-menu is required, and `value` must be another command table or the name of another command table.

When `type:` is `#"divider"`, some sort of a dividing line is displayed in the menu at that point. If a string is supplied using the `options:` init-keyword, it will be drawn as the divider instead of a line. If the look and feel provided by the underlying window system has no corresponding concept, `#"divider"` items may be ignored. When `type:` is `#"divider"`, `value:` is ignored.

The `accelerator:` and `mnemonic:` init-keywords let you specify a keyboard accelerator and mnemonic for the menu item.

Operations	<code>add-command-table-menu-item</code> <code>menu-item-accelerator</code> <code>menu-item-mnemonic</code> <code>menu-item-name</code> <code>menu-item-options</code> <code>menu-item-type</code> <code>menu-item-value</code>
------------	---

See also	<code>add-command-table-menu-item</code> , page 623
----------	---

	<i>Generic function</i>
<b>command-table-name</b>	
Summary	Returns the name of the specified command table.
Signature	<code>command-table-name</code> <code>command-table =&gt; name</code>
Arguments	<code>command-table</code> An instance of type <code>&lt;command-table&gt;</code> .

Values	<i>name</i>	An instance of type <object>.
Description	Returns the name of <i>command-table</i> , as defined by the <code>name:</code> init-keyword for <command-table>.	
See also	<command-table>, page 634 command-table-menu, page 637	

## command-undoable? *Generic function*

Summary	Returns true if the specified command is undoable.
Signature	<code>command-undoable? command =&gt; undoable?</code>
Arguments	<i>command</i> An instance of type <command>.
Values	<i>undoable?</i> An instance of type <boolean>.
Description	Returns true if <i>command</i> is undoable, that is, there is a specified command that the user can choose (for instance, by choosing <b>Edit &gt; Undo</b> ) that will reverse the effects of command.
See also	<code>undo-command</code> , page 750

## complete-from-generator *Generic function*

Summary	Completes a string based on a generated list of completions.
Signature	<code>complete-from-generator string generator delimiters #key action predicate =&gt; string success object nmatches completions</code>
Arguments	<i>string</i> An instance of type <string>.

	<i>generator</i>	An instance of type <function>.
	<i>delimiters</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;character&gt;)</code> .
	<i>action</i>	An instance of type <code>one-of(#"complete", #"complete-limited", #"complete-maximal", #"completions", #"apropos-completions")</code> . Default value <code>#"complete"</code> .
	<i>predicate</i>	An instance of type <code>false-or(&lt;function&gt;)</code> . Default value <code>#f</code> .
Values	<i>string</i>	An instance of type <code>false-or(&lt;string&gt;)</code> .
	<i>success</i>	An instance of type <boolean>.
	<i>object</i>	An instance of type <object>.
	<i>nmatches</i>	An instance of type <integer>.
	<i>completions</i>	An instance of type <sequence>.
Description		<p>Completes <i>string</i> chunk-wise against a list of possibilities derived from <i>generator</i>, using the specified <i>delimiters</i> to break both <i>string</i> and the generated possibilities into chunks. This function is identical to <code>complete-from-sequence</code>, except that the list of possibilities is derived from the <i>generator</i> function, rather than passed explicitly. The <i>generator</i> is a function of two arguments: the string to be completed and a continuation co-routine to call that performs the completion. It should call the continuation with two arguments: the completion string and an object.</p> <p>The argument <i>predicate</i> (if supplied) is applied to filter out unwanted objects.</p> <p>The function returns five values: the completed string (if there is one), whether or not the completion successfully matched, the object associated with the completion, the number of things that matched, and (if <i>action</i> is <code>#"completions"</code>) a sequence of possible completions.</p>

The *action* argument can take any of the following values:

- #"**complete**"      Completes the input as much as possible, except that if the user's input exactly matches one of the possibilities, the shorter possibility is returned as the result, even if it is a left substring of another possibility.
- #"**complete-limited**"      Completes the input up to the next partial delimiter.
- #"**complete-maximal**"      Completes the input as much as possible.
- #"**completions**" or #"**apropos-completions**"  
Returns a sequence of the possible completions.

#### Example

```
complete-from-generator
  ("th", method (string, completer)
    for (b in #["one", "two", "three", "four"])
      completer(b, b)
    end
  end method, #' ', '-'))
```

#### See also

[complete-from-sequence](#), page 642

## complete-from-sequence

## Generic function

**Summary**      Completes a string based on a list of possible completions.

**Signature**

```
complete-from-sequence string possibilities delimiters
  #key action predicate label-key value-key
  => string success object nmatches completions
```

**Arguments**

<i>string</i>	An instance of type < <i>string</i> >.
---------------	--

<i>possibilities</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;string&gt;)</code> .
----------------------	--

	<i>delimiters</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;character&gt;)</code> .
	<i>action</i>	An instance of type <code>one-of(#"complete", #"complete-limited", #"complete-maximal", #"completions", #"apropos-completions")</code> . Default value <code>#"complete"</code> .
	<i>predicate</i>	An instance of type <code>false-or(&lt;function&gt;)</code> . Default value <code>#f</code> .
	<i>label-key</i>	An instance of type <code>&lt;function&gt;</code> . Default value <code>first</code> .
	<i>value-key</i>	An instance of type <code>&lt;function&gt;</code> . Default value <code>second</code> .
Values	<i>string</i>	An instance of type <code>false-or(&lt;string&gt;)</code> .
	<i>success</i>	An instance of type <code>&lt;boolean&gt;</code> .
	<i>object</i>	An instance of type <code>&lt;object&gt;</code> .
	<i>nmatches</i>	An instance of type <code>&lt;integer&gt;</code> .
	<i>completions</i>	An instance of type <code>&lt;sequence&gt;</code> .
Description	<p>Completes <i>string</i> chunk-wise against the list of <i>possibilities</i>, using the specified <i>delimiters</i> to break both <i>string</i> and the strings in <i>possibilities</i> into chunks.</p> <p>The <i>label-key</i> and <i>value-key</i> arguments are used to extract the completion string and object from the entries in <i>possibilities</i>, and <i>predicate</i> (if supplied) is applied to filter out unwanted objects.</p> <p>The function returns five values: the completed string (if there is one), whether or not the completion successfully matched, the object associated with the completion, the number of things that matched, and (if <i>action</i> is <code>#"completions"</code>) a sequence of possible completions.</p> <p>The <i>action</i> argument can take any of the following values:</p>	

**`#"complete"`** Completes the input as much as possible, except that if the user's input exactly matches one of the possibilities, the shorter possibility is returned as the result, even if it is a left substring of another possibility.

**`#"complete-limited"`** Completes the input up to the next partial delimiter.

**`#"complete-maximal"`** Completes the input as much as possible.

**`#"completions" OR #"apropos-completions"`**

Returns a sequence of the possible completions.

**Example**

```
complete-from-sequence("s w ma",
                      #[ "one fish two fish",
                        "red fish blue fish",
                        "single white male",
                        "on beyond zebra"],
                      #[ ' ', '-' ],
                      label-key: identity,
                      value-key: identity)
```

**See also**

**`complete-from-generator`**, page 640

**compute-next-page***Generic function***Summary**

Returns the next page in the specified wizard frame.

**Signature**

**`compute-next-page dialog => next-page`**

**Arguments**

**`dialog`** An instance of type `<wizard-frame>`.

**Values**

**`next-page`** An instance of type `false-or(<sheet>)`.

**Description**

Returns the next page in *dialog*, which must be a wizard.

See also      [compute-previous-page](#), page 645  
[<wizard-frame>](#), page 751

## **compute-previous-page** *Generic function*

Summary      Returns the previous page in the specified wizard frame.

Signature      `compute-previous-page dialog => prev-page`

Arguments      `dialog`      An instance of type `<wizard-frame>`.

Values      `prev-page`      An instance of type `false-or(<sheet>)`.

Description      Returns the previous page in `dialog`, which must be a wizard.

See also      [compute-next-page](#), page 644  
[<wizard-frame>](#), page 751

## **contain** *Generic function*

Summary      Creates and returns a frame containing the specified object.

Signature      `contain object #rest initargs #key own-thread? #all-keys => sheet frame`

Arguments      `object`      An instance of type `type-union(<sheet>, <class>, <frame>)`.  
`initargs`      Instances of type `<object>`.

Values      `sheet`      An instance of type `<sheet>`.  
`frame`      An instance of type `<frame>`.

**Description** Creates and returns a frame containing *object*. This function is intended to be used as a convenience function when testing sections of code in development; you are note recommended to use it in your final source code. The function wraps a set of DUIM objects in a frame and displays them on screen, without you needing to worry about the creation, management, or display of frames on the computer screen. The `contain` function is most useful when testing code interactively using the Dylan Interactor.

If `own-thread?` is `#t`, then the window that is created by `contain` runs in its own thread. If not supplied, `own-thread?` is `#f`.

Consider the following expression that calls `contain`:

```
contain(make(<button>));
```

This is equivalent to the fuller expression:

```
begin
  let frame = make(<simple-frame>,
                    title: "container",
                    layout: make(<button>));
  start-frame(frame);
end;
```

As can be seen, when testing short pieces of code interactively in the environment, the former section of code is easier to use than the latter.

**Example** Assigning the result of a `contain` expression allows you to manipulate the DUIM objects being contained interactively, as shown in the example below.

You should assume the following code is typed into the Dylan Interactor, and that each expression is evaluated by pressing the RETURN key at the points indicated.

```

*g* := contain
  (make
    (<list-box>,
     items: #("One", "Two", "Three"),
     label-key:
       method (symbol) as-lowercase
         (as(<string>, symbol))
       end));RETURN

gadget-items(*g*);RETURN

```

As you would expect, evaluating the call to `gadget-items` returns the following result:

```
#("one", "two", "three")
```

In a similar way, you can destructively modify the slot values of any contained DUIM objects

<b>current-frame</b>	<i>Function</i>
----------------------	-----------------

<b>current-frame</b>	<i>Function</i>
Summary	Returns the current frame
Signature	<code>current-frame =&gt; frame</code>
Arguments	None
Values	<code>frame</code> An instance of type <code>&lt;frame&gt;</code>
Description	Returns the current frame.

<b>define command-table</b>	<i>Definition macro</i>
-----------------------------	-------------------------

<b>define command-table</b>	<i>Definition macro</i>
Summary	Defines a new class of command table with the specified name and properties.
Macro call	<code>define command-table name ({supers}, *) {options} end</code>
Arguments	<code>name</code> A Dylan name <sub>bnf</sub> .

<i>supers</i>	A Dylan name <sub>bnf</sub> .
<i>options</i>	A Dylan body <sub>bnf</sub> .
Values	None.
Description	<p>Defines a new class of command table with the specified name and properties. This macro is equivalent to <code>define class</code>, but with additional options.</p> <p>The <i>supers</i> argument specifies a comma-separated list of command tables from which the command table you are creating should inherit. If you are not explicitly inheriting the behavior of other command tables, then <i>supers</i> should have the value <code>*global-command-table*</code>.</p> <p>Each one of the <i>options</i> supplied describes a command for the command table. This can be either a menu item, a separator, or another command table to be included in the command table. You can supply any number of options. Each option take one of the following forms:</p> <pre>menu-item <b>menu-item-descriptor</b>; include <b>command-table-name</b>; separator;</pre> <p>To add a menu item or menu to a command table, include an option of the following form:</p> <pre>menu-item <b>label</b> = <b>command-function</b>           #key <b>accelerator documentation</b></pre> <p><i>label</i> An instance of &lt;string&gt;. This is the label that appears in the menu.</p> <p><i>command-function</i></p> <p>An instance of <code>type-union(&lt;command&gt;, &lt;command-table&gt;, &lt;function&gt;)</code>. The command function is the callback that is invoked to perform the intended operation for the menu item. Note that this can itself be a command table.</p>

<i>accelerator</i>	An instance of <code>false-or(&lt;gesture&gt;)</code> . Default value: <code>#f</code> . This defines a keyboard accelerator that can be used to invoke <i>command-function</i> in preference to the menu item itself.
<i>documentation</i>	An instance of <code>false-or(&lt;string&gt;)</code> . Default value: <code>#f</code> . This specifies a documentation string for the menu item that can be used to provide online help to the user. For menu items, documentation strings are usually displayed in the status bar of your application, when the mouse pointer is placed over the menu item itself.

To add a separator to a menu, just include the following option at the point you want the separator to appear:

```
separator;
```

To include another command table in the current table, include the following option at the point you want the command table to appear:

```
include command-table-name;
```

The commands defined in *command-table-name* are added to the current command table at the appropriate point.

- Example** The following example shows how you might create a command table for the standard Windows **File** menu, and how this could be integrated into the menu bar for an application.

The example assumes that the appropriate command functions have already been defined for each command in the command table.

```

define command-table
  *file-menu-command-table* (*global-command-table*)
  menu-item "New..."           = frame-new-file,
  accelerator:
    make-keyboard-gesture(#"n", #"control"),
    documentation: "Creates a new document."
  menu-item "Open..."          = frame-open-file,
  accelerator:
    make-keyboard-gesture(#"o", #"control"),
    documentation: "Opens an existing document.";
  menu-item "Close"            = frame-close-file,
  documentation: "Closes an open document.";
  separator;
  include *save-files-command-table*;
  separator;
  menu-item "Exit"
    = make(<command>,
           function: exit-frame);
end command-table *file-menu-command-table*;

define command-table
  *application-command-table* (*global-command-table*)
  menu-item "File"             = *file-menu-command-table*;
  menu-item "Edit"              = *edit-menu-command-table*;
  menu-item "View"              = *view-menu-command-table*;
  menu-item "Windows"           = *windows-menu-command-table*;
  menu-item "Help"               = *help-menu-command-table*;
end command-table *application-command-table*;
```

See also      `*global-command-table*`, page 725

## **define frame** *Definition macro*

Summary      Defines a new class of frame with the specified properties.

Macro call    `define frame name ({supers},*) {slots-panes-options} end`

Arguments    *name*                  A Dylan name<sub>bnf</sub>.

<i>supers</i>	A Dylan name <sub>bnf</sub> .
<i>slots-panes-options</i>	A Dylan body <sub>bnf</sub> .
Values	None.
Description	<p>Defines a new class of frame called <i>name</i> with the specified properties. This macro is equivalent to <code>define class</code>, but with additional options.</p> <p>The <i>supers</i> argument lets you specify any classes from which the frame you are creating should inherit. You must include at least one concrete frame class, such as <code>&lt;simple-frame&gt;</code> or <code>&lt;dialog-frame&gt;</code>.</p> <p>The <i>slots-panes-options</i> supplied describe the state variables of the frame class; that is, the total composition of the frame. This includes, but is not necessarily limited to, any panes, layouts, tool bar, menus, and status bar contained in the frame. You can specify arbitrary slots in the definition of the frame. You may specify any of the following:</p> <ul style="list-style-type: none"> <li>• A number of slots for defining per-instance values of the frame state.</li> <li>• A number of named panes. Each pane defines a sheet of some sort.</li> <li>• A single layout.</li> <li>• A tool bar.</li> <li>• A status bar.</li> <li>• A menu bar.</li> <li>• A command table.</li> <li>• A number of sequential pages for inclusion in a multi-page frame such as a wizard or property dialog.</li> </ul> <p><b>Note:</b> If the frame has a menu bar, either define the menu bar and its panes, or a command table, but not both. See the discussion below for more details.</p>

The syntax for each of these options is described below.

The `slot` option allows you to define any slot values that the new frame class should allow. This option has the same syntax as slot specifiers in `define class`, allowing you to define init-keywords, required init-keywords, init-functions and so on for the frame class.

For each of the remaining options, the syntax is as follows:

***option name (owner) body;***

The argument `option` is the name of the option used, taken from the list described below, `name` is the name you assign to the option for use within your code, `owner` is the owner of the option, usually the frame itself, and `body` contains the definition of value returned by the option.

`pane` specifies a single pane in the frame. The default is `#f`, meaning that there is no single pane. This is the simplest way to define a pane hierarchy.

`layout` specifies the layout of the frame. The default is to lay out all of the named panes in horizontal strips. The value of this option must evaluate to an instance of a layout.

`command-table` defines a command table for the frame. The default is to create a command table with the same name as the frame. The value of this option must evaluate to an instance of `<command-table>`.

`menu-bar` is used to specify the commands that will in the menu bar of the frame. The default is `#t`. If used, it typically specifies the top-level commands of the frame. The value of this option can evaluate to any of the following:

- |                  |  |
|------------------|--|
| <code>#f</code>  | The frame has no menu bar.   |
| <code>#t,</code> | The menu bar for the frame is defined by the value of the <code>command-table</code> option. |

### A command table

The menu bar for the frame is defined by this command table.

**A body of code** This is interpreted the same way as the `menu-item` options to `define command-table`.

`disabled-commands` is used to specify a list of command names that are initially disabled in the application frame. The default is `#[]`. The set of enabled and disabled commands can be modified via `command-enabled?-setter`.

`tool-bar` is used to specify a tool bar for the frame. The default is `#f`. The value of this option must evaluate to an instance of `<tool-bar>`.

`top-level` specifies a function that executes the top level loop of the frame. It has as its argument a list whose first element is the name of a function to be called to execute the top-level loop. The function must take at least one argument, which is the frame itself. The rest of the list consists of additional arguments to be passed to the function.

`icon` specifies an `<image>` to be used in the window decoration for the frame. This icon may be used in the title bar of the frame, or when the frame is iconized, for example.

`geometry` specifies the geometry for the frame.

`pages` is used to define the pages of a wizard or property frame. This evaluates to a list of pages, each of which can be defined as panes within the frame definition itself. For example:

```
define frame <wizard-type> (<wizard-frame>)
  ...
  pages (frame)
    vector(frame.page-1, frame.page-2, frame.page-3);
end frame <wizard-type>
```

The `name`, `supers`, and slot arguments are not evaluated. The values of each of the options are evaluated.

## Example

```

define frame <multiple-values-dialog> (<dialog-frame>
    pane label-pane (frame)
        make(<option-box>, items: #("&Red", "&Green",
                                    "&Blue"));
    pane check-one (frame)
        make(<check-button>, label: "Check box test text");
    pane check-two (frame)
        make(<check-button>, label: "Check box test text");
    pane radio-box (frame)
        make(<radio-box>,
            items: #("Option &1", "Option &2",
                    "Option &3", "Option &4"),
            orientation: #"vertical");
    pane first-group-box (frame)
        grouping ("Group box", max-width: $fill)
            vertically (spacing: 4)
                make(<label>, label: "Label:");
                horizontally (spacing: 4,
                               y-alignment: #"center")
                    frame.label-pane;
                    make(<button>, label: "Button");
            end;
        frame.check-one;
        frame.check-two;
    end;
    end;
    pane second-group-box (frame)
        grouping ("Group box", max-width: $fill)
            frame.radio-box
        end;
    layout (frame)
        vertically (spacing: 4)
            frame.first-group-box;
            frame.second-group-box;
        end;
    end frame <multiple-values-dialog>;

```

## See also

[`<simple-frame>`](#), page 746

[`<wizard-frame>`](#), page 751

**deiconify-frame***Generic function*

**Summary** Displays a frame that has previously been iconified on screen.

**Signature** `deiconify-frame frame => ()`

**Arguments** `frame` An instance of type `<frame>`.

**Values** None

**Description** Displays a frame that has previously been iconified on screen.

**Example** The following example creates and displays a simple frame, then iconifies it and deiconifies it.

```
define variable *frame* =
    make(<simple-frame>, title: "A frame",
        layout: make(<button>));

start-frame(*frame*);
iconify-frame(*frame*);
deiconify-frame(*frame*);
```

**See also** `destroy-frame`, page 655

`exit-frame`, page 692

`frame-icon`, page 707

`iconify-frame`, page 725

**destroy-frame***Generic function*

**Summary** Unmaps the specified frame and destroys it.

**Signature** `destroy-frame frame => ()`

**Arguments** `frame` An instance of type `<frame>`.

Values	None
Description	Unmaps <i>frame</i> from the screen and destroys it. Generally, you should not need to call this function explicitly, since <code>exit-frame</code> performs all necessary operations in the correct order, including calling <code>destroy-frame</code> if the <i>destroy?</i> argument to <code>exit-frame</code> is true.
See also	<a href="#">deiconify-frame</a> , page 655 <a href="#">exit-frame</a> , page 692 <a href="#">&lt;frame-destroyed-event&gt;</a> , page 702 <a href="#">iconify-frame</a> , page 725 <a href="#">lower-frame</a> , page 727 <a href="#">raise-frame</a> , page 741

## dialog-apply-button *Generic function*

Summary	Returns the Apply button in the specified dialog.	
Signature	<code>dialog-apply-button dialog =&gt; apply-button</code>	
Arguments	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>apply-button</code>	An instance of type <code>false-or(&lt;button&gt;)</code> .
Description	Returns the Apply button in <i>dialog</i> . As well as having OK and Cancel buttons, many dialogs also have an Apply button that lets the user apply the changes that have been made in the dialog, without removing the dialog from the screen itself.	
See also	<a href="#">dialog-cancel-button</a> , page 660 <a href="#">dialog-apply-button-setter</a> , page 657 <a href="#">dialog-apply-callback</a> , page 657	

**dialog-help-button**, page 674

## dialog-apply-button-setter *Generic function*

Summary	Specifies the Apply button in the specified dialog.	
Signature	<b>dialog-apply-button-setter</b> <i>apply-button dialog =&gt; apply-button</i>	
Arguments	<i>apply-button</i>	An instance of type <b>false-or(&lt;button&gt;)</b> .
	<i>dialog</i>	An instance of type <b>&lt;dialog-frame&gt;</b> .
Values	<i>apply-button</i>	An instance of type <b>false-or(&lt;button&gt;)</b> .
Description	Specifies the Apply button in <i>dialog</i> . As well as having OK and Cancel buttons, many dialogs also have an Apply button that lets the user apply the changes that have been made in the dialog, without removing the dialog from the screen itself.	
See also	<a href="#">dialog-cancel-button</a> , page 660 <a href="#">dialog-apply-button</a> , page 656 <a href="#">dialog-apply-callback</a> , page 657 <a href="#">dialog-help-button</a> , page 674	

## dialog-apply-callback *Generic function*

Summary	Returns the callback invoked when the Apply button is clicked in the specified dialog.
Signature	<b>dialog-apply-callback</b> <i>dialog =&gt; callback</i>
Arguments	<i>dialog</i> An instance of type <b>&lt;dialog-frame&gt;</b> .

Values	<code>callback</code>	An instance of type <code>false-or(type-non-functional&gt;, &lt;command&gt;)</code> .
Description	Returns the callback invoked when the Apply button is clicked in <i>dialog</i> . As well as having OK and Cancel buttons, many dialogs also have an Apply button that lets the user apply the changes that have been made in the dialog, without removing the dialog from the screen itself.	
<p><b>Note:</b> If you supply <code>#f</code> as the callback, then the button does not appear.</p>		
See also	<a href="#">dialog-cancel-button</a> , page 660 <a href="#">dialog-apply-button</a> , page 656 <a href="#">dialog-apply-button-setter</a> , page 657 <a href="#">dialog-help-button</a> , page 674	

## dialog-back-button *Generic function*

Summary	Returns the Back button in the specified multi-page dialog.	
Signature	<code>dialog-back-button dialog =&gt; back-button</code>	
Arguments	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>back-button</code>	An instance of type <code>false-or(&lt;button&gt;)</code> .
Description	Returns the Back button in <i>dialog</i> . This is most useful in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.	
See also	<a href="#">dialog-back-button-setter</a> , page 659 <a href="#">dialog-back-callback</a> , page 659	

**dialog-exit-button**, page 664

**dialog-help-button**, page 674

## dialog-back-button-setter *Generic function*

**Summary** Specifies the Back button in the specified multi-page dialog.

**Signature** **dialog-back-button-setter** *back-button dialog => back-button*

**Arguments** *back-button* An instance of type <button>.

*dialog* An instance of type <dialog-frame>.

**Values** *back-button* An instance of type <button>.

**Description** Specifies the Back button in *dialog*. This is most useful in wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.

**See also** **dialog-back-button**, page 658

**dialog-back-callback**, page 659

**dialog-exit-button-setter**, page 665

**dialog-help-button**, page 674

## dialog-back-callback *Generic function*

**Summary** Returns the callback invoked when the Back button is clicked in the specified multi-page dialog.

**Signature** **dialog-apply-callback** *dialog => callback*

**Arguments** *dialog* An instance of type <dialog-frame>.

Values	<code>callback</code>	An instance of type <code>false-or(type-non-functional&gt;, &lt;command&gt;)</code> .
Description	Returns the callback invoked when the Back button is clicked in <i>dialog</i> . This is most useful in wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.	<b>Note:</b> If you do not explicitly supply this callback, the previous page in the sequence for the multi-page dialog is displayed when the Back button is clicked. Specifying your own callback gives you flexibility in describing how the user can navigate through the sequence of pages in the dialog.
See also	<a href="#">dialog-back-button</a> , page 658 <a href="#">dialog-back-button-setter</a> , page 659 <a href="#">dialog-exit-callback</a> , page 666 <a href="#">dialog-help-button</a> , page 674	

## dialog-cancel-button *Generic function*

Summary	Returns the Cancel button in the specified dialog.
Signature	<code>dialog-cancel-button dialog =&gt; cancel-button</code>
Arguments	<code>dialog</code> An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>cancel-button</code> An instance of type <code>false-or(&lt;button&gt;)</code> .
Description	Returns the Cancel button in <i>dialog</i> .
See also	<a href="#">dialog-cancel-button-setter</a> , page 661 <a href="#">dialog-cancel-callback</a> , page 662

**dialog-exit-button**, page 664

**dialog-help-button**, page 674

## dialog-cancel-button-setter

## Generic function

**Summary** Specifies the Cancel button in the specified dialog.

**Signature** `dialog-cancel-button-setter cancel-button dialog  
=> cancel-button`

**Arguments** `cancel-button` An instance of type <button>.   
`dialog` An instance of type <dialog-frame>.

**Values** `cancel-button` An instance of type <button>.

**Description** Specifies the Cancel button in *dialog*.

**Example** In the following example, a simple dialog frame is created, and then its cancel button is redefined before the dialog is displayed on screen.

```
define variable *dialog*
  = make(<dialog-frame>,
        exit-button?: #t,
        cancel-button?: #t,
        help-callback:
          method (gadget)
            notify-user (format-to-string
                         ("Here is some help",
                          gadget))
          end);
dialog-cancel-button-setter
  (make(<push-button>, label: "No",
        activate-callback: cancel-dialog,
        max-width: $fill), *dialog*);

start-frame(*dialog*);
```

**See also** [dialog-cancel-button](#), page 660

**dialog-cancel-callback**, page 662  
**dialog-exit-button-setter**, page 665  
**dialog-help-button-setter**, page 675

**dialog-cancel-callback** *Generic function*

Summary      Returns the function invoked when the cancel button is clicked in the specified dialog.

Signature     **dialog-cancel-callback** *dialog* => *callback*

Arguments    *dialog*       An instance of type **<dialog-frame>**.

Values       *callback*     An instance of type **false-or(type-union(<command>, <function>))**.

Library      **duim-frames**

Module        **duim-frames**

Description    Returns the function invoked when the cancel button is clicked in *dialog*. This defaults to **cancel-dialog**.

See also     [cancel-dialog](#), page 629

[dialog-cancel-button](#), page 660

[dialog-cancel-button-setter](#), page 661

[dialog-exit-callback](#), page 666

[dialog-help-callback](#), page 676

**dialog-cancel-callback-setter** *Generic function*

Summary	Sets the function invoked when the cancel button is clicked in the specified dialog.	
Signature	<code>dialog-cancel-callback-setter callback dialog =&gt; callback</code>	
Arguments	<i>callback</i>	An instance of type <code>false-or(&lt;command&gt;, &lt;function&gt;)</code> . Default value: <code>cancel-dialog</code> .
	<i>dialog</i>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<i>callback</i>	An instance of type <code>false-or(&lt;command&gt;, &lt;function&gt;)</code> .
Library	<code>duim-frames</code>	
Module	<code>duim-frames</code>	
Description	Sets the function invoked when the cancel button is clicked in <i>dialog</i> .	
See also	<a href="#">dialog-cancel-button</a> , page 660 <a href="#">dialog-cancel-button-setter</a> , page 661 <a href="#">dialog-exit-callback</a> , page 666 <a href="#">dialog-help-callback</a> , page 676	

**dialog-current-page** *Generic function*

Summary	Returns the current page in the specified multi-page dialog.	
Signature	<code>dialog-current-page dialog =&gt; page</code>	
Arguments	<i>dialog</i>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<i>page</i>	An instance of type <code>false-or(&lt;page&gt;)</code> .

Library	<b>duim-frames</b>
Module	<b>duim-frames</b>
Description	Returns the current page in <i>dialog</i> .
See also	<b>dialog-current-page-setter</b> , page 664

## **dialog-current-page-setter** *Generic function*

Summary	Sets the current page in the specified multi-page dialog.
Signature	<b>dialog-current-page-setter</b> <i>page dialog =&gt; page</i>
Arguments	<i>page</i> An instance of type < <b>page</b> >. <i>dialog</i> An instance of type < <b>dialog-frame</b> >.
Values	<i>page</i> An instance of type < <b>page</b> >.
Library	<b>duim-frames</b>
Module	<b>duim-frames</b>
Description	Sets the current page in <i>dialog</i> .
See also	<b>dialog-current-page</b> , page 663

## **dialog-exit-button** *Generic function*

Summary	Returns the Exit button in the specified dialog.
Signature	<b>dialog-exit-button</b> <i>dialog =&gt; exit-button</i>
Arguments	<i>dialog</i> An instance of type < <b>dialog-frame</b> >.

Values	<b><i>exit-button</i></b>	An instance of type <code>false-or(&lt;button&gt;)</code> .
Description	Returns the Exit button in <i>dialog</i> . The Exit button is commonly found in multi-page dialogs, where the user is given the option to exit the sequence at any point (as well as navigate through the sequence using Next and Back buttons).	
See also		<a href="#">dialog-cancel-button</a> , page 660 <a href="#">dialog-exit-button-setter</a> , page 665 <a href="#">dialog-exit-enabled?</a> , page 668 <a href="#">dialog-exit-callback</a> , page 666 <a href="#">dialog-help-button</a> , page 674

## dialog-exit-button-setter *Generic function*

Summary	Specifies the Exit button in the specified dialog.	
Signature	<b><code>dialog-exit-button-setter</code></b> <b><i>exit-button</i></b> <b><i>dialog</i></b> => <b><i>exit-button</i></b>	
Arguments	<b><i>exit-button</i></b>	An instance of type <code>&lt;button&gt;</code> .
	<b><i>dialog</i></b>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<b><i>exit-button</i></b>	An instance of type <code>&lt;button&gt;</code> .
Description	Sets the Exit button in <i>dialog</i> . The Exit button is commonly found in multi-page dialogs, where the user is given the option to exit the sequence at any point (as well as navigate through the sequence using Next and Back buttons).	
Example	In the following example, a simple dialog frame is created, and then its exit button is redefined before the dialog is displayed on screen.	

```

define variable *dialog*
  = make(<dialog-frame>,
        exit-button?: #t,
        cancel-button?: #t,
        help-callback:
          method (gadget)
            notify-user (format-to-string
                          ("Here is some help",
                           gadget))
          end);

dialog-exit-button-setter
  (make(<push-button>, label: "Yes",
        activate-callback: exit-dialog,
        max-width: $fill), *dialog*);

start-frame(*dialog*);

```

See also      [dialog-cancel-button-setter](#), page 661  
               [dialog-exit-button](#), page 664  
               [dialog-exit-enabled?](#), page 668  
               [dialog-exit-callback](#), page 666  
               [dialog-help-button-setter](#), page 675

**dialog-exit-callback***Generic function*

Summary	Returns the callback invoked when the Exit button is clicked in the specified dialog.	
Signature	<b>dialog-exit-callback</b> <i>dialog</i> => <i>callback</i>	
Arguments	<i>dialog</i>	An instance of type < <b>dialog-frame</b> >.
Values	<i>callback</i>	An instance of type <b>false-or(type-union(&lt;command&gt;, &lt;function&gt;))</b> . Default value: <b>exit-dialog</b> .
Library	<b>duim-frames</b>	

Module	<b>duim-frames</b>
Description	Returns the callback invoked when the Exit button is clicked in <i>dialog</i> . The Exit button is commonly found in multi-page dialogs, where the user is given the option to exit the sequence at any point (as well as navigate through the sequence using Next and Back buttons).
See also	<a href="#">dialog-cancel-callback</a> , page 662 <a href="#">dialog-exit-button</a> , page 664 <a href="#">dialog-exit-button-setter</a> , page 665 <a href="#">dialog-exit-callback-setter</a> , page 667 <a href="#">dialog-help-callback</a> , page 676

## dialog-exit-callback-setter *Generic function*

Summary	Sets the callback invoked when the Exit button is clicked in the specified dialog.	
Signature	<b>dialog-exit-callback</b> <i>callback dialog =&gt; callback</i>	
Arguments	<i>callback</i>	An instance of type <b>false-or(type-union(&lt;command&gt;, &lt;function&gt;))</b> .
	<i>dialog</i>	An instance of type <b>&lt;dialog-frame&gt;</b> .
Values	<i>callback</i>	An instance of type <b>false-or(type-union(&lt;command&gt;, &lt;function&gt;))</b> .
Library	<b>duim-frames</b>	
Module	<b>duim-frames</b>	

Description	Sets the callback invoked when the Exit button is clicked in <i>dialog</i> . The Exit button is commonly found in multi-page dialogs, where the user is given the option to exit the sequence at any point (as well as navigate through the sequence using Next and Back buttons).  If you do not supply this callback, then the default behavior is to quit the dialog when the Exit button is clicked. This is normally the action that you will want. Specifying your own callback gives you flexibility in describing other actions to be performed when the dialog is exited. In addition, supplying <code>#f</code> means that no Exit button is displayed at all.
See also	<a href="#">dialog-cancel-callback-setter</a> , page 663 <a href="#">dialog-exit-button</a> , page 664 <a href="#">dialog-exit-button-setter</a> , page 665 <a href="#">dialog-exit-callback</a> , page 666 <a href="#">dialog-help-callback</a> , page 676

## dialog-exit-enabled?

*Generic function*

Summary	Returns true if the Exit button has been enabled for the specified dialog.	
Signature	<code>dialog-exit-enabled? dialog =&gt; enabled?</code>	
Arguments	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>enabled?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns true if the Exit button has been enabled for <i>dialog</i> . The Exit button is commonly found in multi-page dialogs, where the user is given the option to exit the sequence at any point (as well as navigate through the sequence using Next and Back buttons).	

See also [dialog-exit-button](#), page 664  
[dialog-exit-button-setter](#), page 665  
[dialog-exit-enabled?-setter](#), page 669  
[dialog-exit-callback](#), page 666

## dialog-exit-enabled?-setter *Generic function*

Summary	Enables or disables the Exit button for the specified dialog.				
Signature	<code>dialog-exit-enabled?-setter enabled? dialog =&gt; enabled?</code>				
Arguments	<table> <tr> <td><code>enabled?</code></td><td>An instance of type <code>&lt;boolean&gt;</code>.</td></tr> <tr> <td><code>dialog</code></td><td>An instance of type <code>&lt;dialog-frame&gt;</code>.</td></tr> </table>	<code>enabled?</code>	An instance of type <code>&lt;boolean&gt;</code> .	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
<code>enabled?</code>	An instance of type <code>&lt;boolean&gt;</code> .				
<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .				
Values	<code>enabled?</code> An instance of type <code>&lt;boolean&gt;</code> .				
Description	Enables or disables the Exit button for <i>dialog</i> . The Exit button is commonly found in multi-page dialogs, where the user is given the option to exit the sequence at any point (as well as navigate through the sequence using Next and Back buttons).				
Example	In this example, a dialog is created, and then its exit button is disabled. When displayed on the screen, the exit button is grayed out and you cannot click on it.				

```
define variable *dialog* =
    make(<dialog-frame>,
        exit-button?: #t,
        cancel-button?: #t,
        help-callback:
            method (gadget)
                notify-user
                    (format-to-string
                        ("Here is some help",
                            gadget))
            end);
dialog-exit-enabled?-setter(#f, *dialog*);
```

```
start-frame(*dialog*);
```

See also      [dialog-exit-button](#), page 664  
[dialog-exit-button-setter](#), page 665  
[dialog-exit-enabled?](#), page 668  
[dialog-exit-callback](#), page 666

## <dialog-frame>

*Open abstract instantiable class*

Summary      The class of dialog frames.

Superclasses    <[simple-frame](#)>

Init-keywords    **mode:**      An instance of type `one-of("modal", "#\"modeless\"", "#\"system-modal\"")`. Default value: `#\"modal\"`.  
**exit-callback:** An instance of type `false-or(type-union(<command>, <function>))`. Default value: `exit-dialog`.

**exit-button:** An instance of type `false-or(<button>)`. Default value: `#f`.

**exit-enabled?:** An instance of type `<boolean>`. Default value: `#t`.

**cancel-callback**

An instance of type `false-or(type-union(<command>, <function>))`. Default value: `cancel-dialog`.

**cancel-button:** An instance of type `false-or(<button>)`. Default value: `#f`.

**help-callback:** An instance of type `false-or(type-union(<command>, <function>))`. Default value: `#f`.

**help-button:** An instance of type `false-or(<button>)`.  
 Default value: `#f`.

**exit-buttons-position:**

An instance of type `one-of(#"top", "#"bottom", #"left", #"right")`. Default value: `#"bottom"`.

**pages:** An instance of type `false-or(<sequence>)`.  
 Default value: `#f`.

**page-changed-callback:**

An instance of type `false-or(<function>)`.  
 Default value: `#f`.

Description	The class of dialog frames. These frames let you create dialog boxes for use in your applications. All buttons in a dialog frame are automatically made the same size, and are placed at the bottom of the dialog by default. When at the bottom of the dialog, buttons are right-aligned.
-------------	--



**Figure 9.2** A typical dialog

By default, all dialogs are modal, that is, when displayed, they take over the entire application thread, preventing the user from using any other part of the application until the dialog has been removed from the screen. To create a modeless dialog (that is, one that can remain displayed on the screen while the user interacts with the application in other ways) you should set the `mode:` keyword to `#"modeless"`. Note, however, that you should not normally need to do this:

if you need to create a modeless dialog, then you should consider using a normal DUIM frame, rather than a dialog frame.

The init-keywords `exit-button:`, and `cancel-button:` specify the exit and cancel buttons in the dialog. The user clicks on the exit button to dismiss the dialog and save any changes that have been made as a result of editing the information in the dialog. The user clicks on the cancel button in order to dismiss the dialog and discard any changes that have been made.

In addition, the `exit-callback:` and `cancel-callback:` init-keywords specify the callback that is invoked when the Exit or Cancel buttons in the dialog are clicked on. These both default to the appropriate function for each button, but you have the flexibility to specify an alternative if you wish. If you do not require a Cancel button in your dialog, specify `cancel-callback: #f`. Similarly, specify `exit-callback: #f` if you do not require an Exit button.

All dialogs should have an exit button, and most dialogs should have a cancel button too. You should only omit the cancel button in cases when the information being displayed in the dialog cannot be changed by the user. For example, a dialog containing an error message can have only an exit button, but any dialog that contains information the user can edit should have both exit and cancel buttons.

Two init-keywords are available for each button so that a given button may be specified for a particular dialog, but need only be displayed in certain circumstances. This lets you define subtly different behavior in different situations.

The `exit-enabled?:` init-keyword is used to specify whether the exit button on the dialog is enabled or not. If `#f`, then the exit button is displayed on the dialog, but it is grayed out.

The **help-button:** init-keyword specifies the help button in the dialog. Note that, in contrast to the exit and cancel buttons, specifying the button gadget to use in a dialog determines its presence in the dialog: it is not possible to define a help button and then only display it in certain circumstances. You are strongly encouraged to provide a help button in all but the most trivial dialogs.

The **help-callback:** init-keyword defines a callback function that is invoked when the help button is clicked. This should normally display a context-sensitive help topic from the help file supplied with the application, although you might also choose to display an alert box with the relevant information.

The **exit-buttons-position:** init-keyword defines the position in the dialog that the exit and cancel buttons occupy (and any other standard buttons, if they have been specified). By default, buttons are placed where the interface guidelines for the platform recommend, and this position is encouraged in most interface design guidelines. Usually, this means that buttons are placed at the bottom of the dialog. Less commonly, buttons may also be placed on the right side of the dialog. Buttons are not normally placed at the top or on the left of the dialog, though this is possible if desired.

The **pages:** init-keyword is used for multi-page dialogs such as property frames and wizard frames. If used, it should be a sequence of elements, each of which evaluates to an instance of a page.

The **page-changed-callback:** is a callback function that is invoked when a different page in a multi-page dialog is displayed.

Operations	The following operations are exported from the <b>DUIM-Frames</b> module.
------------	---

```

cancel-dialog dialog-cancel-button
dialog-cancel-button-setter dialog-cancel-callback
dialog-cancel-callback-setter dialog-exit-button
dialog-exit-button-setter dialog-exit-callback
dialog-exit-callback-setter dialog-exit-enabled?
dialog-exit-enabled?-setter dialog-exit-callback
dialog-exit-callback-setter dialog-help-button
dialog-help-button-setter dialog-help-callback exit-
dialog start-dialog

```

**Example**

The following example creates and displays a simple dialog that contains only an exit button, cancel button, and help button, and assigns a callback to the help button.

```

define variable *dialog*
  = make(<dialog-frame>,
        exit-button?: #t,
        cancel-button?: #t,
        help-callback:
          method (gadget)
            notify-user (format-to-string
                         ("Here is some help",
                          gadget))
          end);
  start-frame(*dialog*);

```

**See also**

[cancel-dialog](#), page 629  
[exit-dialog](#), page 691  
[<property-frame>](#), page 739  
[<simple-frame>](#), page 746  
[<wizard-frame>](#), page 751

**dialog-help-button***Generic function***Summary**

Returns the Help button in the specified dialog.

Signature	<code>dialog-help-button dialog =&gt; help-button</code>	
Arguments	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>help-button</code>	An instance of type <code>false-or(&lt;button&gt;)</code> .
Description	Returns the Help button in <code>dialog</code> . Many dialogs contain a Help button that, when clicked, displays a relevant topic from the online help system for the application.	
See also	<a href="#">dialog-cancel-button</a> , page 660 <a href="#">dialog-exit-button</a> , page 664 <a href="#">dialog-help-button-setter</a> , page 675 <a href="#">dialog-help-callback</a> , page 676	

## dialog-help-button-setter *Generic function*

Summary	Specifies the Help button in the specified dialog.	
Signature	<code>dialog-help-button-setter help-button dialog =&gt; help-button</code>	
Arguments	<code>help-button</code>	An instance of type <code>false-or(&lt;button&gt;)</code> .
	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>help-button</code>	An instance of type <code>false-or(&lt;button&gt;)</code>
Description	Specifies the Help button in <code>dialog</code> . Many dialogs contain a Help button that, when clicked, displays a relevant topic from the online help system for the application.	
Example	In the following example, a simple dialog frame is created, and then its help button is redefined before the dialog is displayed on screen.	

```

define variable *dialog*
  = make(<dialog-frame>,
        exit-button?: #t,
        cancel-button?: #t,
        help-callback:
          method (gadget)
            notify-user (format-to-string
                          ("Here is some help",
                           gadget))
          end);

dialog-help-button-setter
  (make(<push-button>, label: "Help Me!",
        activate-callback:
          method (gadget)
            notify-user
              (format-to-string
                ("Here is some help",
                 gadget))
            end);
   max-width: $fill), *dialog*);

start-frame(*dialog*);

```

## See also

[dialog-cancel-button-setter](#), page 661[dialog-exit-button-setter](#), page 665[dialog-help-button](#), page 674[dialog-help-callback](#), page 676**dialog-help-callback***Generic function*

**Summary**      Returns the callback invoked when the Help button is clicked in the specified dialog.

**Signature**      `dialog-help-callback dialog => help-callback`

**Arguments**      `dialog`      An instance of type `<dialog-frame>`.

**Values**      `help-callback`      An instance of type `false-or(type-union(<command>, <function>))`.

Library	<b>duim-frames</b>
Module	<b>duim-frames</b>
Description	Returns the callback invoked when the Help button is clicked in <i>dialog</i> . Many dialogs contain a Help button that, when clicked, displays a relevant topic from the online help system for the application.
	<b>Note:</b> You must specify this callback in order to create a Help button in any dialog. If the callback is #f, then there will be no Help button present in the dialog.
See also	<a href="#">dialog-cancel-callback</a> , page 662 <a href="#">dialog-exit-callback</a> , page 666 <a href="#">dialog-help-button</a> , page 674 <a href="#">dialog-help-button-setter</a> , page 675

## dialog-next-button *Generic function*

Summary	Returns the Next button in the specified multi-page dialog.	
Signature	<b>dialog-next-button</b> <i>dialog</i> => <i>next-button</i>	
Arguments	<i>dialog</i>	An instance of type < <b>dialog-frame</b> >.
Values	<i>next-button</i>	An instance of type <b>false-or(&lt;button&gt;)</b> .
Description	Returns the Next button in <i>dialog</i> . This is most useful in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.	
See also	<a href="#">dialog-back-button</a> , page 658	

**dialog-exit-button**, page 664  
**dialog-next-button-setter**, page 678  
**dialog-next-callback**, page 678

**dialog-next-button-setter** *Generic function*

Summary	Specifies the Next button in the specified multi-page dialog.
Signature	<b>dialog-next-button-setter</b> <i>next-button dialog =&gt; next-button</i>
Arguments	<i>next-button</i> An instance of type <b>false-or(&lt;button&gt;)</b> . <i>dialog</i> An instance of type <b>&lt;dialog-frame&gt;</b> .
Values	<i>next-button</i> An instance of type <b>false-or(&lt;button&gt;)</b> .
Description	Specifies the Next button in <i>dialog</i> . This is most useful in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.
See also	<b>dialog-back-button-setter</b> , page 659 <b>dialog-exit-button</b> , page 664 <b>dialog-next-button</b> , page 677 <b>dialog-next-callback</b> , page 678

**dialog-next-callback** *Generic function*

Summary	Returns the callback invoked when the Next button is clicked in the specified multi-page dialog.
Signature	<b>dialog-apply-callback</b> <i>dialog =&gt; callback</i>

Arguments	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>callback</code>	An instance of type <code>false-or(type-union(&lt;command&gt;, &lt;function&gt;))</code> .
Description		<p>Returns the callback invoked when the Next button is clicked in <code>dialog</code>. This is most useful in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.</p> <p><b>Note:</b> If you do not explicitly supply this callback, the next page in the sequence for the multi-page dialog is displayed when the Next button is clicked. Specifying your own callback gives you flexibility in describing how the user can navigate through the sequence of pages in the dialog.</p> <p>The default value for this callback is <code>move-to-next-page</code>.</p>
See also		<p><code>dialog-back-button</code>, page 658</p> <p><code>dialog-exit-callback</code>, page 666</p> <p><code>dialog-next-button</code>, page 677</p> <p><code>dialog-next-button-setter</code>, page 678</p> <p><code>move-to-next-page</code>, page 736</p>

**dialog-next-enabled?***Generic function*

Summary	Returns true if the Next button has been enabled for the specified multi-page dialog.
Signature	<code>dialog-next-enabled? dialog =&gt; enabled?</code>
Arguments	<code>dialog</code>

Values	<i>enabled?</i>	An instance of type <boolean>.
Description	Returns true if the Next button has been enabled for <i>dialog</i> . This button is most useful in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.	
See also	<a href="#"><code>&lt;dialog-frame&gt;</code>, page 670</a> <a href="#"><code>dialog-next-button</code>, page 677</a> <a href="#"><code>dialog-next-button-setter</code>, page 678</a> <a href="#"><code>dialog-next-enabled?-setter</code>, page 680</a> <a href="#"><code>dialog-next-callback</code>, page 678</a>	

<b>dialog-next-enabled?-setter</b>		<i>Generic function</i>
Summary	Enables or disables the Next button for the specified multi-page dialog.	
Signature	<code>dialog-next-enabled?-setter enabled? dialog =&gt; enabled?</code>	
Arguments	<i>enabled?</i>	An instance of type <boolean>.
	<i>dialog</i>	An instance of type < <code>dialog-frame</code> >.
Values	<i>enabled?</i>	An instance of type <boolean>.
Description	Enables or disables the Next button for <i>dialog</i> . This button is most useful in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.	

It is useful to be able to enable and disable the Next button at any point in order to ensure that the user supplies all necessary information before proceeding to the next page of the dialog. You can do this by testing to see if the information on the page has been specified with `dialog-page-complete?`, and then enabling or disabling the Next button as appropriate.

### Example

See also `dialog-next-button`, page 677  
`dialog-next-button-setter`, page 678  
`dialog-next-callback`, page 678  
`dialog-next-enabled?`, page 679

	<i>Generic function</i>
Summary	Returns the next page in sequence for the specified multi-page dialog.
Signature	<code>dialog-next-page dialog =&gt; next-page</code>
Arguments	<code>dialog</code> An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>next-page</code> An instance of type <code>false-or(&lt;page&gt;)</code> .
Description	Returns the next page in sequence for <code>dialog</code> . This is for use in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.

The default method for the Next button in *dialog* uses the value of this function. When the Next button is clicked, the current page is set to the next logical page in the sequence, but you are free to dynamically change it as the state of the dialog changes.

See also

**dialog-next-button**, page 677  
**dialog-next-button-setter**, page 678  
**dialog-next-callback**, page 678  
**dialog-next-page-setter**, page 682  
**dialog-previous-page**, page 687

## dialog-next-page-setter

*Generic function*

**Summary**      Specifies the next page in sequence for the specified multi-page dialog.

**Signature**      **dialog-next-page-setter** *next-page dialog => next-page*

**Arguments**      *next-page*      An instance of type **false-or(<page>)**.  
                      *dialog*      An instance of type **<dialog-frame>**.

**Values**      *next-page*      An instance of type **false-or(<page>)**.

**Description**      Specifies the next page in sequence for *dialog*. This is for use in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.

The default method for the Next button in *dialog* uses the value of this function. When the Next button is clicked, the current page is set to the next logical page in the sequence, but you are free to dynamically change it as the state of the dialog changes.

See also	<a href="#">dialog-next-button</a> , page 677 <a href="#">dialog-next-button-setter</a> , page 678 <a href="#">dialog-next-callback</a> , page 678 <a href="#">dialog-next-page</a> , page 681 <a href="#">dialog-previous-page-setter</a> , page 688
----------	---

## **dialog-page-changed-callback** *Generic function*

Summary	Returns the page-changed callback of the specified multi-page dialog.	
Signature	<code>dialog-page-changed-callback dialog =&gt; callback</code>	
Arguments	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>callback</code>	An instance of type <code>false-or(type-union(&lt;command&gt;, &lt;function&gt;))</code> .
Description	Returns the page-changed-callback of <i>dialog</i> . This is the callback function used to test whether the information in the current page of <i>dialog</i> has changed. This callback is useful when using multi-page dialogs, as a test that can be performed before the next page of the dialog is displayed.	
See also	<a href="#">&lt;dialog-frame&gt;</a> , page 670 <a href="#">dialog-page-changed-callback-setter</a> , page 684 <a href="#">&lt;property-frame&gt;</a> , page 739	

&lt;wizard-frame&gt;, page 751

**dialog-page-changed-callback-setter** *Generic function*

**Summary** Sets the page-changed callback of the specified multi-page dialog.

**Signature** `dialog-page-changed-callback-setter callback dialog => callback`

**Arguments** `callback` An instance of type `false-or(type-union(<command>, <function>))`.

`dialog` An instance of type `<dialog-frame>`.

**Values** `callback` An instance of type `false-or(type-union(<command>, <function>))`.

**Description** Sets the page-changed-callback of `dialog`. This is the callback function used to test whether the information in the current page of `dialog` has changed. This callback is useful when using multi-page dialogs, as a test that can be performed before the next page of the dialog is displayed.

**See also** `<dialog-frame>`, page 670

`dialog-page-changed-callback`, page 683

`<property-frame>`, page 739

`<wizard-frame>`, page 751

**dialog-page-complete?** *Generic function*

**Summary** Returns true if all the information required on the current page of the specified multi-page dialog has been specified.

Signature	<code>dialog-page-complete? dialog =&gt; complete?</code>	
Arguments	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>complete?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	<p>Returns true if all the information required on the current page in <code>dialog</code> has been specified by the user. This generic function has two uses:</p> <ul style="list-style-type: none"> <li>• It can be used within wizards to test whether all the necessary information has been supplied, before moving on to the next page of the wizard.</li> <li>• It can be used within property pages to test whether all the necessary information has been supplied, before allowing the user to apply any changes.</li> </ul>	
See also	<code>dialog-page-complete?-setter</code> , page 685	

## **dialog-page-complete?-setter** *Generic function*

Summary	Sets the slot that indicates all the information required on the current page of the specified multi-page dialog has been specified.	
Signature	<code>dialog-page-complete? complete? dialog =&gt; complete?</code>	
Arguments	<code>complete?</code>	An instance of type <code>&lt;boolean&gt;</code> .
	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>complete?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	<p>Sets the slot that indicates all the information required on the current page in <code>dialog</code> has been specified by the user. This generic function has two uses:</p>	

- It can be used within wizards to indicate that the necessary information has been supplied, so that the next page of the wizard can be displayed safely.
  - It can be used within property pages to indicate that the necessary information has been supplied, so that the user can apply any changes.

See also [dialog-page-complete?](#), page 684

## dialog-pages *Generic function*

**Summary** Returns the pages of the specified multi-page dialog.

Signature **dialog-pages** *dialog => pages*

Arguments      *dialog*                  An instance of type `<dialog-frame>`.

Values            *pages*            An instance of type `limited(<sequence>, of; <page>)`.

Description Returns the pages of *dialog*. Each of the items in sequence is an instance of `<page>`.

See also <dialog-frame>, page 670

**dialog-pages-setter**, page 686

<property-frame>, page 739

<wizard-frame>, page 751

## dialog-pages-setter *Generic function*

**Summary** Sets the pages of the specified multi-page dialog.

Signature      **dialog-pages-setter** *pages dialog => pages*

Arguments	<i>pages</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;page&gt;)</code> .
	<i>dialog</i>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<i>pages</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;page&gt;)</code> .
Description		Sets the pages of <i>dialog</i> . Each of the items in sequence must be an instance of <code>&lt;page&gt;</code> .
See also		<p><code>&lt;dialog-frame&gt;</code>, page 670</p> <p><code>dialog-pages</code>, page 686</p> <p><code>&lt;property-frame&gt;</code>, page 739</p> <p><code>&lt;wizard-frame&gt;</code>, page 751</p>

## dialog-previous-page *Generic function*

Summary	Returns the previous page in sequence for the specified multi-page dialog.	
Signature	<code>dialog-previous-page dialog =&gt; previous-page</code>	
Arguments	<i>dialog</i>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<i>previous-page</i>	An instance of type <code>false-or(&lt;page&gt;)</code> .
Description	Returns the previous page in sequence for <i>dialog</i> . This is for use in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.	

The default method for the Back button in *dialog* uses the value of this function. When the Back button is clicked, the current page is set to the previous logical page in the sequence, but you are free to dynamically change it as the state of the dialog changes.

## See also

`dialog-back-button`, page 658  
`dialog-back-button-setter`, page 659  
`dialog-back-callback`, page 659  
`dialog-next-page`, page 681  
`dialog-previous-page-setter`, page 688

**dialog-previous-page-setter***Generic function*

Summary	Specifies the previous page in sequence for the specified multi-page dialog.	
Signature	<code>dialog-previous-page-setter previous-page dialog =&gt; previous-page</code>	
Arguments	<code>previous-page</code>	An instance of type <code>false-or(&lt;page&gt;)</code> .
	<code>dialog</code>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<code>previous-page</code> An instance of type <code>false-or(&lt;page&gt;)</code> .	
Description	Specifies the previous page in sequence for <i>dialog</i> . This is for use in multi-page dialogs such as property frames and wizard frames, which typically have Back and Next buttons that let the user navigate forward and backward through the sequence of pages that comprise the dialog.	

The default method for the Back button in *dialog* uses the value of this function. When the Back button is clicked, the current page is set to the previous logical page in the sequence, but you are free to dynamically change it as the state of the dialog changes.

See also [dialog-back-button](#), page 658  
[dialog-back-button-setter](#), page 659  
[dialog-back-callback](#), page 659  
[dialog-next-page-setter](#), page 682  
[dialog-previous-page](#), page 687

## **display-progress-note** *Generic function*

Summary	Displays the specified progress note.				
Signature	<code>display-progress-note framem progress-note =&gt; ()</code>				
Arguments	<table border="0"> <tr> <td><i>framem</i></td> <td>An instance of type &lt;<b>frame-manager</b>&gt;.</td> </tr> <tr> <td><i>progress-note</i></td> <td>An instance of type &lt;<b>progress-note</b>&gt;.</td> </tr> </table>	<i>framem</i>	An instance of type < <b>frame-manager</b> >.	<i>progress-note</i>	An instance of type < <b>progress-note</b> >.
<i>framem</i>	An instance of type < <b>frame-manager</b> >.				
<i>progress-note</i>	An instance of type < <b>progress-note</b> >.				
Values	None				
Description	Displays the specified <i>progress-note</i> in the frame managed by <i>framem</i> .				

## **doc-command-table-menu-commands**

### **event-destroy-frame?** *Generic function*

Summary	Returns information about the frame was destroyed in the specified event.
---------	---

Signature	<code>event-destroy-frame? event =&gt; destroyed?</code>	
Arguments	<code>event</code>	An instance of type <code>&lt;frame-exit-event&gt;</code> .
Values	<code>destroyed?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Returns information about the frame was destroyed in <code>event</code> .	
See also	<code>&lt;frame-exit-event&gt;</code> , page 705	

## **event-status-code** *Generic function*

Summary	Returns the status code of the specified event.	
Signature	<code>event-status-code event =&gt; code</code>	
Arguments	<code>event</code>	An instance of type <code>&lt;frame-exited-event&gt;</code> .
Values	<code>code</code>	An instance of type <code>false-or(&lt;integer&gt;)</code> .
Description	Returns the status code of <code>event</code> .	
See also	<code>&lt;frame-exited-event&gt;</code> , page 704	

## **execute-command** *Generic function*

Summary	Executes a command for the specified frame.	
Signature	<code>execute-command command frame =&gt; #rest values</code>	
Arguments	<code>command</code>	An instance of type <code>&lt;command&gt;</code> .
	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>values</code>	Instances of type <code>&lt;object&gt;</code> .

Description	Executes <i>command</i> for <i>frame</i> . The values returned are those values returned as a result of evaluating the command function of <i>command</i> .
-------------	---

## exit-dialog *Generic function*

Summary	Exits the specified dialog.				
Signature	<code>exit-dialog <i>dialog</i> #key <i>destroy?</i> =&gt; ()</code>				
Arguments	<table> <tr> <td><i>dialog</i></td><td>An instance of type &lt;dialog-frame&gt;.</td></tr> <tr> <td><i>destroy?</i></td><td>An instance of type &lt;boolean&gt;. Default value: #t.</td></tr> </table>	<i>dialog</i>	An instance of type <dialog-frame>.	<i>destroy?</i>	An instance of type <boolean>. Default value: #t.
<i>dialog</i>	An instance of type <dialog-frame>.				
<i>destroy?</i>	An instance of type <boolean>. Default value: #t.				
Values	None				
Description	<p>Exits <i>dialog</i>, recording any changes to the information displayed in the dialog that have been made by the user.</p> <p>This is the default callback used for the exit button in a dialog. This is the button that is typically labeled <b>OK</b>.</p> <p>If <i>destroy?</i> is #t, then <i>dialog</i> is destroyed.</p>				
Example	<p>The following example defines a button, *yes-button*, that calls <code>exit-dialog</code> as its activate-callback. This button is then used in a dialog that simply replaces the standard exit button for the newly defined dialog. Note that the example assumes the existence of a similar *no-button* to replace the cancel button.</p> <pre>define variable *yes-button*   = make(&lt;push-button&gt;, label: "Yes",         activate-callback: exit-dialog,         max-width: \$fill);</pre>				

```

define variable *dialog*
  = make(<dialog-frame>,
        exit-button?: #f,
        cancel-button?: #f,
        layout: vertically
          (x-alignment: #"center",
           y-spacing: 5)
        make(<label>,
              label: "Here is a label");
        horizontally (x-spacing: 2)
          *yes-button*;
          *no-button*;
        end
      end);

start-frame(*dialog*);
```

See also      [cancel-dialog](#), page 629  
[<dialog-frame>](#), page 670  
[start-dialog](#), page 748

## **exit-frame** *Generic function*

Summary	Unmaps the specified frame destroying it required.	
Signature	<b>exit-frame</b> <i>frame</i> #key <i>destroy?</i> => ()	
Arguments	<i>frame</i>	An instance of type <a href="#">&lt;frame&gt;</a> .
	<i>destroy?</i>	An instance of type <a href="#">&lt;boolean&gt;</a> . Default value: #t.
Values	None	
Description	<p>Unmaps <i>frame</i>, removing the associated sheet and its children from the screen. If <i>destroy?</i> is true, then the frame is destroyed completely, via a call to <a href="#">destroy-frame</a>.</p> <p>If <i>destroy?</i> is #t, then dialog is destroyed.</p>	

**Example** The following example creates a simple frame, then displays it and exits it. You should run this code in the interactor, pressing the RETURN key at the points indicated.

```
define variable *frame* =
  make(<simple-frame>, title: "A frame",
       layout: make(<button>));RETURN

start-frame(*frame*);RETURN

exit-frame(*frame*);RETURN
```

**See also**

- `destroy-frame`, page 655
- `frame-can-exit?`, page 700
- `<frame-exited-event>`, page 704
- `<frame-exit-event>`, page 705
- `frame-mapped?-setter`, page 712
- `start-frame`, page 748

	<i>Function</i>										
<b>find-frame</b>											
<b>Summary</b>	Returns a frame of the specified type, creating one if necessary.										
<b>Signature</b>	<code>find-frame frame-class #rest initargs #key create? activate? own-thread? port frame-manager test #all-keys =&gt; frame</code>										
<b>Arguments</b>	<table border="0"> <tr> <td><i>frame-class</i></td><td>An instance of type <code>&lt;object&gt;</code>.</td></tr> <tr> <td><i>initargs</i></td><td>An instance of type <code>&lt;object&gt;</code>.</td></tr> <tr> <td><i>create?</i></td><td>An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</td></tr> <tr> <td><i>activate?</i></td><td>An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</td></tr> <tr> <td><i>own-thread?</i></td><td>An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#t</code>.</td></tr> </table>	<i>frame-class</i>	An instance of type <code>&lt;object&gt;</code> .	<i>initargs</i>	An instance of type <code>&lt;object&gt;</code> .	<i>create?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .	<i>activate?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .	<i>own-thread?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
<i>frame-class</i>	An instance of type <code>&lt;object&gt;</code> .										
<i>initargs</i>	An instance of type <code>&lt;object&gt;</code> .										
<i>create?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .										
<i>activate?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .										
<i>own-thread?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .										

	<i>port</i>	An instance of type < <b>port</b> >.
	<i>frame-manager</i>	An instance of type < <b>frame-manager</b> >.
	<i>test</i>	An instance of type < <b>function</b> >. Default value: <b>identity</b> .
Values	<i>frame</i>	An instance of type < <b>frame</b> >.
Description		<p>This function creates a frame of the specified type if one does not already exist, and then runs it, possibly in its own thread. If one already exists, then it is selected.</p> <p>The <i>frame-class</i> argument specifies the class of frame that is being searched for. By default, if a match is not found, then an instance of this class will be created.</p> <p>The <i>init-args</i> supplied are the slot values that should be passed to the instance of frame-class. Either an existing frame must be found that has the specified slot values, or a new one will be created.</p> <p>If <i>create?</i> is <code>#t</code>, then a new frame will not be created if it does not already exist.</p> <p>If <i>own-thread?</i> is true, the frame will run in its own thread if one is created.</p> <p>The <i>port</i> and <i>frame-manager</i> arguments specify a port and frame manager which control the frame being searched for, or under the control of which a new frame should be created.</p> <p>If desired, you can supply a <i>test</i> which must evaluate to true for a frame to match successfully.</p>
See also		<a href="#">&lt;frame&gt;</a> , page 694

<b>&lt;frame&gt;</b>	<i>Open abstract class</i>
Summary	The base class of all frames.

Superclasses    `<object>`

Init-keywords

- `owner:`    An instance of type `false-or(<frame>)`.  
Default value: `#f`.
- `mode:`    An instance of type `one-of(#"modeless",  
#"modal", #"system-modal")`. Default  
value: `"modeless"`.
- `default-button:`  
An instance of type `false-or(<button>)`.  
Default value: `#f`.
- `x:`    An instance of type `<integer>`.
- `y:`    An instance of type `<integer>`.
- `width:`    An instance of type `<integer>`.
- `height:`    An instance of type `<integer>`.
- `disabled-commands:`  
An instance of type `<sequence>`.
- `top-level-sheet:`  
An instance of type `false-or(<sheet>)`.  
Default value: `#f`.
- `layout:`    An instance of type `<layout>`.
- `icon:`    An instance of type `false-or(<image>)`.
- `title:`    An instance of type `false-or(<string>)`.  
Default value: `#f`.
- `calling-frame:` An instance of type `<frame>`.
- `state:`    An instance of type `one-of(#"detached",  
#"unmapped", #"mapped", #"iconified")`.  
Default value: `"detached"`.
- `thread:`    An instance of type `false-or(<thread>)`.  
Default value: `#f`.

<b>event-queue:</b>	An instance of type <code>false-or(&lt;event-queue&gt;)</code> . Default value: <code>#f</code> .
<b>input-focus:</b>	An instance of type <code>false-or(&lt;sheet&gt;)</code> . Default value: <code>#f</code> .
<b>foreground:</b>	An instance of type <code>false-or(&lt;ink&gt;)</code> .
<b>background:</b>	An instance of type <code>false-or(&lt;ink&gt;)</code> .
<b>text-style:</b>	An instance of type <code>false-or(&lt;text-style&gt;)</code> .
<b>palette:</b>	An instance of type <code>false-or(&lt;palette&gt;)</code> . Default value: <code>#f</code> .
<b>document:</b>	An instance of type <code>false-or(&lt;object&gt;)</code> . Default value: <code>#f</code> .
<b>resource-id:</b>	An instance of type <code>false-or(&lt;integer&gt;)</code> .
<b>resizable?:</b>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
<b>fixed-width?:</b>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#f</code> .
<b>fixed-height?:</b>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#f</code> .
<b>Description</b>	<p>The class of all frames.</p> <p>The <code>owner</code>: init-keyword is the parent of the frame.</p> <p>The <code>mode</code>: init-keyword lets you specify the mode for the frame. By default, frames are modeless, that is, they do not take over control of the whole application when they are mapped, and the user can interact with other frames in the application normally. Modal frames, on the other hand, behave like a <code>&lt;dialog-frame&gt;</code>, restricting the user's interaction with other frames in the application until the modal frame has been dismissed.</p>

The `default-button`: init-keyword is used to specify which button is the default action for the frame. The default button is usually the one whose callback is invoked by pressing the RETURN key.

The `x:`, `y:`, `width:` and `height:` init-keywords lets you specify the initial size and position of the frame. The position is specified using `x:` and `y:`, which represent the number of pixels from the top left corner of the screen, and the `width:` and `height:` init-keywords specify the initial size of the frame.

The `title`: init-keyword is used to specify a title for the frame.

The `state`: init-keyword is used to specify the initial state of the frame. This describes whether the frame is mapped, whether it is iconified, and so on. By default, new frames are detached.

By default, new frames run in their own thread. If desired, a frame can be run in an existing thread by setting the `thread`: init-keyword to the thread object concerned. For more information about threads, see the manual *Library Reference: Core Features*.

As with threads, new frame run in their own event-queue by default. To run the frame in an existing event-queue, use the `event-queue`: init-keyword.

You can specify which sheet in the frame initially has the input-focus using the `input-focus`: init-keyword. The input-focus dictates where information can be typed by default.

The `foreground`, `background`, and `text-style`: init-keywords describes the colors and fonts used in the frame.

Specify a palette for the frame using the `palette`: init-keyword.

Specify a resource-id for the frame using the `resource-id:` init-keyword. This is a platform-specific ID or determining which resource to use to fill in a frame.

The `resizable?::`, `fixed-width?::`, and `fixed-height?::` init-keywords let you specify whether or not the user can resize the frame. If `resizable?::` is `#t`, then the frame can be resized in either direction; if it is `#f`, then it cannot be resized at all. In addition, if `resizable?::` is `#t`, and one of `fixed-width?::` or `fixed-height?::` is also `#t`, then the frame is resizable, but is fixed in the appropriate direction. For example, if `resizable?::` is `#t` and `fixed-height?::` is also `#t`, then only the width of the frame can be resized.

Operations	The following operations are exported from the <code>DUIM-Frames</code> module.
------------	---

```
apply-in-frame call-in-frame command-enabled?  
command-enabled?-setter deiconify-frame destroy-  
frame execute-command exit-frame frame?  
frame-accelerators frame-accelerators-setter frame-  
can-exit? frame-default-button frame-default-button-  
setter frame-event-queue frame-icon frame-icon-setter  
frame-input-focus frame-input-focus-setter frame-  
mapped? frame-mapped?-setter frame-mode frame-owner  
frame-palette frame-palette-setter frame-position  
frame-size frame-state frame-thread frame-title frame-  
title-setter iconify-frame lower-frame layout-frame  
raise-frame redo-command set-frame-position set-  
frame-size layout-frame undo-command
```

The following operations are exported from the `DUIM-Sheets` module.

```
beep display force-display frame-manager handle-event
```

The following operations are exported from the `DUIM-DCs` module.

```
default-background default-foreground default-text-
style find-color port queue-event synchronize-display
top-level-sheet
```

## frame?

*Generic function*

Summary	Returns true if the specified object is a frame.	
Signature	<b>frame? object =&gt; frame?</b>	
Arguments	<i>object</i>	An instance of type <object>.
Values	<i>frame?</i>	An instance of type <boolean>.
Description	Returns true if <i>object</i> is a frame. Use this generic function to test that an object is a frame before carrying out frame-related operations on it.	
See also	<a href="#">current-frame</a> , page 647 <a href="#">&lt;frame&gt;</a> , page 694	

## frame-accelerators

*Generic function*

Summary	Returns the keyboard accelerators defined for the specified frame.	
Signature	<b>frame-accelerators frame =&gt; accelerators</b>	
Arguments	<i>frame</i>	An instance of type <frame>.
Values	<i>accelerators</i>	An instance of type <b>false-or(limited(&lt;sequence&gt;, of: &lt;gesture&gt;))</b> .
Description	Returns the keyboard accelerators defined for <i>frame</i> .	

See also [frame-accelerators-setter](#), page 700

frame-accelerators-setter	Generic function				
Summary	Defines the keyboard accelerators for the specified frame.				
Signature	<code>frame-accelerators accelerators frame =&gt; accelerators</code>				
Arguments	<table><tr><td><code>accelerators</code></td><td>An instance of type <code>false-or(limited(&lt;sequence&gt;, of: &lt;gesture&gt;))</code>.</td></tr><tr><td><code>frame</code></td><td>An instance of type <code>&lt;frame&gt;</code>.</td></tr></table>	<code>accelerators</code>	An instance of type <code>false-or(limited(&lt;sequence&gt;, of: &lt;gesture&gt;))</code> .	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
<code>accelerators</code>	An instance of type <code>false-or(limited(&lt;sequence&gt;, of: &lt;gesture&gt;))</code> .				
<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .				
Values	<code>accelerators</code> An instance of type <code>false-or(limited(&lt;sequence&gt;, of: &lt;gesture&gt;))</code> .				
Description	Defines the keyboard accelerators for <i>frame</i> .				
See also	<code>frame-accelerators</code> , page 699				

frame-can-exit?	<i>Open generic function</i>
Summary	Returns true if the specified frame can be exited dynamically.
Signature	<code>frame-can-exit? frame =&gt; can-exit?</code>
Arguments	<code>frame</code> An instance of type < <code>frame</code> >.
Values	<code>can-exit?</code> An instance of type < <code>boolean</code> >.
Description	Returns true if <code>frame</code> can be exited dynamically. You can add methods to this generic function in order to allow the user to make a dynamic decision about whether a frame should exit.

**Example**

```
define method frame-can-exit?
  (frame :: <abstract-test-frame>) =>
    (can-exit? :: <boolean>)
    notify-user("Really exit?", 
      frame: frame,
      style: #"question")
end method frame-can-exit?;
```

**See also**

[exit-frame](#), page 692

**frame-command-table**

*Generic function*

**Summary**

Returns the command table associated with the specified frame.

**Signature**

**frame-command-table** *frame* => *command-table*

**Arguments**

*frame* An instance of type <**frame**>.

**Values**

*command-table* An instance of type **false-or(<command-table>)**.

**Description**

Returns the command table associated with *frame*.

**See also**

[frame-command-table-setter](#), page 701

**frame-command-table-setter**

*Generic function*

**Summary**

Specifies the command table associated with the specified frame.

**Signature**

**frame-command-table-setter** *command-table* *frame*
=> *command-table*

**Arguments**

*command-table* An instance of type <**command-table**>.

*frame* An instance of type <**frame**>.

Values	<i>command-table</i> An instance of type < <b>command-table</b> >.
Description	Specifies the command table associated with <i>frame</i> .
See also	<b>frame-command-table</b> , page 701

**<frame-created-event>** *Sealed instantiable class*

Summary	The class of events that indicate a frame has been created.
Superclasses	< <b>frame-event</b> >
Init-keywords	None.
Description	The class of events that indicate a frame has been created. An instance of this class is distributed to the frame when it is created. Only one of these events is passed during the lifetime of any frame.
Operations	None.
See also	< <b>frame-destroyed-event</b> >, page 702 < <b>frame-exited-event</b> >, page 704

**<frame-destroyed-event>** *Sealed instantiable class*

Summary	The class of events that indicate a frame has been destroyed.
Superclasses	< <b>frame-event</b> >
Init-keywords	None.

Description	The class of events that indicate a frame has been destroyed. An instance of this class is distributed to the frame when it is destroyed. Only one of these events is passed during the lifetime of any frame.
Operations	None.
See also	<a href="#">destroy-frame</a> , page 655 <a href="#">&lt;frame-created-event&gt;</a> , page 702 <a href="#">&lt;frame-exited-event&gt;</a> , page 704

**frame-default-button***Generic function*

Summary	Returns the default button associated with the specified frame.
Signature	<b>frame-default-button</b> <i>frame</i> => <i>default-button</i>
Arguments	<i>frame</i> An instance of type <a href="#">&lt;frame&gt;</a> .
Values	<i>default-button</i> An instance of type <a href="#">false-or(&lt;button&gt;)</a> .
Description	Returns the default button associated with <i>frame</i> .
See also	<a href="#">frame-default-button-setter</a> , page 703

**frame-default-button-setter***Generic function*

Summary	Sets the default button associated with the specified frame.
Signature	<b>frame-default-button-setter</b> <i>default-button</i> <i>frame</i> => <i>default-button</i>
Arguments	<i>default-button</i> An instance of type <a href="#">false-or(&lt;button&gt;)</a> .

	<i>frame</i>	An instance of type < <b>frame</b> >.
Values	<i>default-button</i>	An instance of type <b>false-or(&lt;button&gt;)</b> .
Description		Sets the default button associated with <i>frame</i> .
See also		<b>frame-default-button</b> , page 703

## **frame-event-queue** *Generic function*

Summary	Returns the event queue that the specified frame is running in.	
Signature	<b>frame-event-queue</b> <i>frame</i> => <i>event-queue</i>	
Arguments	<i>frame</i>	An instance of type < <b>frame</b> >.
Values	<i>event-queue</i>	An instance of type < <b>event-queue</b> >.
Description		Returns the event queue that <i>frame</i> is running in.
See also	<b>&lt;frame&gt;</b> , page 694	

## **<frame-exited-event>** *Sealed instantiable class*

Summary	The class of events that indicate a frame has been exited.	
Superclasses	<b>&lt;frame-event&gt;</b>	
Init-keywords	<b>status-code:</b>	An instance of type <b>false-or(&lt;integer&gt;)</b> . This class also inherits the <b>frame:</b> init-keyword from its superclass.
Description		

Operations	None.
Example	<p>The class of events that indicate a frame has been exited. An instance of this class is distributed to the frame when it is exited. Only one of these events is passed during the lifetime of any frame.</p> <p>The <code>status-code:</code> init-keyword is used to pass a status code, if desired. This code can be used to pass the reason that the frame was exited.</p>
See also	<p><code>&lt;application-exited-event&gt;</code>, page 627</p> <p><code>exit-frame</code>, page 692</p> <p><code>&lt;frame-created-event&gt;</code>, page 702</p> <p><code>&lt;frame-destroyed-event&gt;</code>, page 702</p>

## `<frame-exit-event>` *Sealed instantiable class*

Summary	The class of events distributed when a frame is about to exit.
Superclasses	<code>&lt;frame-event&gt;</code>
Init-keywords	<p><code>destroy-frame?:</code></p> <p>An instance of type <code>&lt;boolean&gt;</code>. Default value: <code>#f</code>.</p>
Description	<p>The class of events distributed when a frame is about to exit. Contrast this with <code>&lt;frame-exited-event&gt;</code>, which is passed after the frame is exited.</p> <p>The default method uses <code>frame-can-exit?</code> to decide whether or not to exit.</p> <p>If <code>destroy-frame?:</code> is <code>#t</code>, then the frame is destroyed.</p>
Operations	None.

See also      `event-destroy-frame?`, page 689  
`frame-can-exit?`, page 700  
`<frame-exited-event>`, page 704

### **<frame-focus-event>**

*Sealed instantiable class*

Summary      The class of events distributed when a frame receives focus.

Superclasses      `<frame-event>`

Init-keywords      None.

Description      The class of events distributed when a frame receives the mouse focus.

Operations      None.

See also      `event-destroy-frame?`, page 689

`frame-can-exit?`, page 700

`<frame-exited-event>`, page 704

### **frame-fixed-height?**

*Generic function*

Summary      Returns true if the height of the specified frame is not resizable.

Signature      `frame-fixed-width? frame => fixed-height?`

Arguments      `frame`      An instance of type `<frame>`.

Values      `fixed-height?`      An instance of type `<boolean>`.

Description      Returns true if the height of `frame` is not resizable.

## Example

See also      **frame-fixed-width?**, page 707  
**frame-resizable?**, page 717

### **frame-fixed-width?** *Generic function*

Summary      Returns true if the width of the specified frame is not resizable.

Signature      **frame-fixed-width? *frame* => *fixed-width?***

Arguments      *frame*      An instance of type <**frame**>.

Values      *fixed-width?*      An instance of type <**boolean**>.

Description      Returns true if the width of *frame* is not resizable.

## Example

See also      **frame-fixed-height?**, page 706  
**frame-resizable?**, page 717

### **frame-icon** *Generic function*

Summary      Returns the icon associated with the specified frame.

Signature      **frame-icon *frame* => *icon***

Arguments      *frame*      An instance of type <**frame**>.

Values      *icon*      An instance of type **false-or(<image>)**.

Description	Returns the icon associated with <i>frame</i> . This is the icon used to represent the frame when it has been iconized. In Windows 95 and Windows NT 4.0, this icon is also visible in the left hand corner of the title bar of the frame when it is not iconized.
See also	<a href="#">deiconify-frame</a> , page 655 <a href="#">frame-icon-setter</a> , page 708 <a href="#">iconify-frame</a> , page 725

**frame-icon-setter** *Generic function*

Summary	Specifies the icon associated with the specified frame.	
Signature	<code>frame-icon-setter icon frame =&gt; icon</code>	
Arguments	<i>icon</i>	An instance of type <code>false-or(&lt;image&gt;)</code> .
	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> .
Values	<i>icon</i> An instance of type <code>false-or(&lt;image&gt;)</code> .	
Description	Specifies the icon associated with <i>frame</i> . This icon is used when the frame is iconified, and in Windows 95 and Windows NT 4.0 is also visible on the left hand side of the title bar of the frame.	
See also	<a href="#">frame-icon</a> , page 707	

**frame-input-focus** *Generic function*

Summary	Returns the sheet in the specified frame that has the input focus.
---------	--

Signature	<code>frame-input-focus frame =&gt; focus</code>	
Arguments	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>focus</code>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Returns the sheet in <code>frame</code> that has the input focus.	
See also	<a href="#">frame-input-focus-setter</a> , page 709	

## frame-input-focus-setter *Generic function*

Summary	Sets which sheet in the specified frame has the input focus.	
Signature	<code>frame-input-focus-setter focus frame =&gt; focus</code>	
Arguments	<code>focus</code>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>focus</code>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .
Description	Sets which sheet in <code>frame</code> has the input focus.	
See also	<a href="#">frame-input-focus</a> , page 708	

## frame-layout *Generic function*

Summary	Returns the layout used in the specified frame.	
Signature	<code>frame-layout frame =&gt; layout</code>	
Arguments	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>layout</code>	An instance of type <code>false-or(&lt;sheet&gt;)</code> .

Description      Returns the layout used in *frame*.

See also      **frame-layout-setter**, page 710

## **frame-layout-setter**

*Generic function*

Summary      Specifies the layout used in the specified frame.

Signature      **frame-layout-setter** *layout frame => layout*

Arguments      *layout*      An instance of type **false-or(<sheet>)**.

*frame*      An instance of type **<frame>**.

Values      *layout*      An instance of type **false-or(<sheet>)**.

Description      Specifies the layout used in *frame*.

See also      **frame-layout**, page 709

## **frame-mapped?**

*Generic function*

Summary      Returns true if the specified frame is mapped.

Signature      **frame-mapped?** *frame => mapped?*

Arguments      *frame*      An instance of type **<frame>**.

Values      *mapped?*      An instance of type **<boolean>**.

Description      Returns true if *frame* is mapped, that is, is currently displayed on-screen. Note that a frame is considered to be mapped if it is anywhere on the screen, even if it is not completely visible because other windows are covering it either partially or completely, or if it is iconized.

**Example** The following example creates a simple frame, then displays it and exits it. In between starting and exiting the frame, `frame-mapped?` is called. You should run this code in the interactor, pressing the RETURN key at the points indicated.

```
define variable *frame* =
  make(<simple-frame>, title: "A frame",
    layout: make(<button>));RETURN

start-frame(*frame*);RETURN

frame-mapped?(*frame*);RETURN
=> #t

exit-frame(*frame*);RETURN

frame-mapped?(*frame*);RETURN
=> #f
```

**See also** `frame-mapped?-setter`, page 712

## <frame-mapped-event>

*Sealed instantiable class*

**Summary** The class of events that indicate a frame has been mapped.

**Superclasses** `<frame-event>`

**Init-keywords** None.

**Description** The class of events that indicate a frame has been mapped, that is, displayed on screen. An instance of this class is distributed whenever a frame is mapped.

**Operations** None.

**Example** The following example defines a method that can inform you when an instance of a class of frame you have defined is mapped.

```

define method handle-event
    (frame :: <my-frame>,
     event :: <frame-mapped-event>) => ()
  notify-user
    (format-to-string("Frame %= mapped", frame))
end method handle-event;

```

See also      `<frame-unmapped-event>`, page 724

## frame-mapped?-setter *Generic function*

Summary	Maps or unmaps the specified frame.	
Signature	<code>frame-mapped?-setter mapped? frame =&gt; mapped?</code>	
Arguments	<code>mapped?</code>	An instance of type <code>&lt;boolean&gt;</code> .
	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>mapped?</code>	An instance of type <code>&lt;boolean&gt;</code> .
Description	Maps or unmaps <i>frame</i> , that is, displays frame on the screen or removes it from the screen, depending on whether <i>mapped?</i> is true or false. Note that a frame is considered to be mapped if it is anywhere on the screen, even if it is not completely visible because other windows are covering it either partially or completely, or if it is iconized.	
Example	The following example creates a simple frame, then displays it and unmaps it using <code>frame-mapped?-setter</code> rather than <code>start-frame</code> and <code>exit-frame</code> . You should run this code in the interactor, pressing the RETURN key at the points indicated.	
	<pre> define variable *frame* =   make(&lt;simple-frame&gt;, title: "A frame",        layout: make(&lt;button&gt;));RETURN  frame-mapped?-setter(#t, *frame*);RETURN </pre>	

```
frame-mapped?-setter(#f, *frame*);RETURN
```

See also      [exit-frame](#), page 692  
[frame-mapped?](#), page 710  
[start-frame](#), page 748

## frame-menu-bar *Generic function*

Summary     Returns the menu bar used in the specified frame.  
 Signature    **frame-menu-bar frame => menu-bar**  
 Arguments    *frame*       An instance of type <[frame](#)>.  
 Values        *menu-bar*     An instance of type [false-or\(<menu-bar>\)](#).  
 Description   Returns the menu bar used in *frame*.  
 See also      [frame-menu-bar-setter](#), page 713

## frame-menu-bar-setter *Generic function*

Summary     Sets the menu bar used in the specified frame.  
 Signature    **frame-menu-bar-setter menu-bar frame => menu-bar**  
 Arguments    *menu-bar*     An instance of type [false-or\(<menu-bar>\)](#).  
                *frame*       An instance of type <[frame](#)>.  
 Values        *menu-bar*     An instance of type [false-or\(<menu-bar>\)](#).  
 Description   Sets the menu bar used in *frame*.  
 See also      [frame-menu-bar](#), page 713

<b>frame-mode</b>	<i>Generic function</i>
Summary	Returns the mode of the specified frame.
Signature	<b>frame-mode frame =&gt; mode</b>
Arguments	frame                          An instance of type < <b>frame</b> >.
Values	mode                          An instance of type one-of(#"modeless", #"modal", #" <b>system-modal</b> ").
Description	<p>Returns the mode of <i>frame</i>. This is the same value as was specified for the <code>mode:</code> init-keyword when the frame was created.</p> <p>If <i>frame</i> is modal, such as a dialog, then it must be dismissed before the user can interact with the user interface of an application (for instance, before a menu can be displayed).</p> <p>If <i>frame</i> is modeless, then the user can interact with its parent frame while the frame is still visible. Typically, the user will move the frame to a convenient position on the screen and continue work, keeping the frame on screen for as long as is desired. For example it is often useful to make the Find dialog box in an application modeless, so that the user can keep it on screen while performing other tasks.</p> <p>If <i>frame</i> is system-modal, then it prevents the user from interacting with <b>any</b> other running applications, such as the Shutdown dialog in Windows 95. System modal frames are rarely used, and should be used with caution.</p> <p><b>Note:</b> You can only set the mode of a frame when it is first created. The mode cannot subsequently be changed.</p>
See also	< <b>frame</b> >, page 694

**frame-owner** *Generic function*

Summary	Returns the controlling frame for the specified frame.	
Signature	<code>frame-owner frame =&gt; owner</code>	
Arguments	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>owner</code>	An instance of type <code>false-or(&lt;frame&gt;)</code> .
Description	Returns the controlling frame for <i>frame</i> . The controlling frame for any hierarchy of existing frames is the one that owns the thread in which the frames are running. Thus, the controlling frame for <i>frame</i> is not necessarily its direct owner: it may be the owner of <i>frame</i> 's owner, and so on, depending on the depth of the hierarchy.	

**frame-palette** *Generic function*

Summary	Returns the palette used in the specified frame.	
Signature	<code>frame-palette frame =&gt; palette</code>	
Arguments	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>palette</code>	An instance of type <code>&lt;palette&gt;</code> .
Description	Returns the palette used in <i>frame</i> .	
See also	<code>frame-palette-setter</code> , page 715	

**frame-palette-setter** *Generic function*

Summary	Sets the palette used in the specified frame.
---------	---

Signature	<code>frame-palette-setter palette frame =&gt; palette</code>	
Arguments	<code>palette</code>	An instance of type <palette>.
	<code>frame</code>	An instance of type <frame>.
Values	<code>palette</code>	An instance of type <palette>.
Description	Sets the palette used in <i>frame</i> .	
See also	<a href="#">frame-palette</a> , page 715	

## frame-position *Generic function*

Summary	Returns the position on the screen of the specified frame.
Signature	<code>frame-position frame =&gt; x y</code>
Arguments	<code>frame</code> An instance of type <frame>.
Values	<code>x</code> An instance of type <integer>. <code>y</code> An instance of type <integer>.
Description	Returns the position on the screen of <i>frame</i> . Coordinates are expressed relative to the top left corner of the screen, measured in pixels.
Example	The following example creates a simple frame, then displays it and tests its position. You should run this code in the interactor, pressing the RETURN key at the points indicated.

```
define variable *frame* =
  make(<simple-frame>, title: "A frame",
       layout: make(<button>));RETURN

start-frame(*frame*);RETURN

frame-position(*frame*);RETURN
```

See also      **frame-size**, page 717  
**frame-state**, page 718  
**set-frame-position**, page 744

## **frame-resizable?** *Generic function*

Summary     Returns true if the specified frame is resizable.  
Signature    **frame-resizable? frame => resizable?**  
Arguments    *frame*       An instance of type <**frame**>.  
Values       *resizable?*    An instance of type <**boolean**>.  
Description   Returns true if *frame* is resizable, that is can have one or both of its width and height modified by the user.

### Example

See also    **frame-fixed-height?**, page 706  
**frame-fixed-width?**, page 707

## **frame-size** *Generic function*

Summary     Returns the size of the specified frame.  
Signature    **frame-size frame => width height**  
Arguments    *frame*       An instance of type <**frame**>.  
Values       *width*        An instance of type <**integer**>.  
              *height*        An instance of type <**integer**>.

Description	Returns the size of <i>frame</i> , measured in pixels.
Example	The following example creates a simple frame, then displays it and tests its size. You should run this code in the interactor, pressing the RETURN key at the points indicated.
	<pre>define variable *frame* =     make(&lt;simple-frame&gt;, title: "A frame",         layout: make(&lt;button&gt;));RETURN  start-frame(*frame*);RETURN  frame-size(*frame*);RETURN</pre>

See also

- `frame-position`, page 716
- `frame-state`, page 718
- `set-frame-size`, page 745

<b>frame-state</b>		<i>Generic function</i>
Summary	Returns the visible state of the specified frame.	
Signature	<code>frame-state frame =&gt; state</code>	
Arguments	<code>frame</code>	An instance of type <code>&lt;frame&gt;</code> .
Values	<code>state</code>	An instance of type <code>one-of(#"detached", "#"unmapped", "#"mapped", "#"iconified", "#"destroyed")</code> .
Description	Returns the visible state of the specified frame. The return value from this function indicates whether frame is currently iconified, whether it is mapped or unmapped, whether it has been destroyed, or whether it has become detached from the thread of which it was a part.	

**Example** The following example creates a simple frame, then displays it and tests its position. You should run this code in the interactor, pressing the RETURN key at the points indicated.

```
define variable *frame* =
  make(<simple-frame>, title: "A frame",
       layout: make(<button>));RETURN

start-frame(*frame*);RETURN

frame-state(*frame*);RETURN
=> #"mapped"
```

**See also** [frame-position](#), page 716  
[frame-size](#), page 717

**frame-status-bar***Generic function*

**Summary** Returns the status bar used in the specified frame.

**Signature** **frame-status-bar** *frame* => *status-bar*

**Arguments** *frame* An instance of type <**frame**>.

**Values** *status-bar* An instance of type **false-or(<status-bar>)**.

**Description** Returns the status bar used in *frame*.

**See also** [frame-status-bar-setter](#), page 719

**frame-status-bar-setter***Generic function*

**Summary** Sets the status bar used in the specified frame.

**Signature** **frame-status-bar-setter** *status-bar* *frame* => *status-bar*

Arguments	<i>status-bar</i>	An instance of type <code>&lt;status-bar&gt;</code> .
	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> .
Values	<i>status-bar</i>	An instance of type <code>false-or(&lt;status-bar&gt;)</code> .
Description	Sets the status bar used in <i>frame</i> .	
See also	<a href="#">frame-status-bar, page 719</a>	

## frame-status-message

## *Open generic function*

Summary	Returns the status message for the specified frame.
Signature	<b>frame-status-message</b> <i>frame</i> => <i>status-message</i>
Arguments	<i>frame</i> An instance of type <b>&lt;frame&gt;</b> .
Values	<i>status-message</i> An instance of type <b>false-or(&lt;string&gt;)</b> .
Description	Returns the status message for <i>frame</i> . This is the label in the status bar for the frame. If the frame has no status bar, or if the label is not set, this function returns false.
See also	<b>frame-status-bar</b> , page 719 <b>frame-status-message-setter</b> , page 720 <b>&lt;status-bar&gt;</b> , page 573

## frame-status-message-setter

## *Generic function*

<b>Summary</b>	Sets the status message for the specified frame.
<b>Signature</b>	<b>frame-status-message</b> <i>status-message</i> <b>frame</b> => <i>status-message</i>

Arguments	<i>status-message</i>	An instance of type <code>false-or(&lt;string&gt;)</code> .
	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> .
Values	<i>status-message</i>	An instance of type <code>false-or(&lt;string&gt;)</code> .
Description		Sets the status message for <i>frame</i> . This is the label in the status bar for the frame. If the frame has no status bar, then attempting to set the label fails silently.
See also		<a href="#">frame-status-bar-setter, page 719</a> <a href="#">frame-status-message, page 720</a> <a href="#">&lt;status-bar&gt;, page 573</a>

## frame-thread *Generic function*

Summary	Returns the thread with which the specified frame is associated.	
Signature	<code>frame-thread frame =&gt; thread</code>	
Arguments	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> .
Values	<i>thread</i>	An instance of type <code>&lt;thread&gt;</code> .
Description		Returns the thread with which <i>frame</i> is associated. For more information about threads, refer to the manual <i>Library Reference: Core Features</i> .

## frame-title *Generic function*

Summary	Returns the title of the specified frame.	
Signature	<code>frame-title frame =&gt; title</code>	

Arguments	<i>frame</i>	An instance of type <frame>.
Values	<i>title</i>	An instance of type false-or(<string>).
Description		Returns the title of <i>frame</i> . If this is #f, then the title bar is removed from the frame, if this is possible. If this is not possible, then a default message is displayed. Whether the title bar can be removed from the frame or not is platform dependent.
See also		<a href="#">frame-title-setter</a> , page 722

## frame-title-setter *Generic function*

Summary	Sets the title of the specified frame.	
Signature	<b>frame-title-setter</b> <i>title frame =&gt; title</i>	
Arguments	<i>title</i>	An instance of type false-or(<string>).
	<i>frame</i>	An instance of type <frame>.
Values	<i>title</i>	An instance of type false-or(<string>).
Description		Sets the title of <i>frame</i> . The title of a frame is displayed in the title bar of the frame. If <i>title</i> is #f, then the platform attempts to remove the title bar from the frame, if possible.
See also		<a href="#">frame-title</a> , page 721

## frame-tool-bar *Generic function*

Summary	Returns the tool bar used in the specified frame.
Signature	<b>frame-tool-bar</b> <i>frame =&gt; tool-bar</i>

Arguments	<i>frame</i>	An instance of type <frame>.
Values	<i>tool-bar</i>	An instance of type <code>false-or(&lt;tool-bar&gt;)</code> .
Description		Returns the tool bar used in <i>frame</i> .
See also		<code>frame-tool-bar-setter</code> , page 723

**frame-tool-bar-setter***Generic function*

Summary	Sets the tool bar used in the specified frame.	
Signature	<code>frame-tool-bar-setter tool-bar frame =&gt; tool-bar</code>	
Arguments	<i>tool-bar</i>	An instance of type <code>false-or(&lt;tool-bar&gt;)</code> .
	<i>frame</i>	An instance of type <frame>.
Values	<i>tool-bar</i>	An instance of type <code>false-or(&lt;tool-bar&gt;)</code> .
Description	Sets the tool bar used in <i>frame</i> .	
See also	<code>frame-tool-bar</code> , page 722	

**frame-top-level***Generic function*

Summary	Returns the top level loop function for the specified frame.	
Signature	<code>frame-top-level frame =&gt; top-level</code>	
Arguments	<i>frame</i>	An instance of type <frame>.
Values	<i>top-level</i>	An instance of type <function>.

Description	Returns the top level loop function for <i>frame</i> . The top level loop function for a frame is the “command loop” for the frame.
	The default method for <code>frame-top-level</code> calls <code>read-event</code> and then <code>handle-event</code> .
See also	<code>handle-event</code> , page 252

**<frame-unmapped-event>***Sealed instantiable class*

Summary      The class of events that indicate a frame has been unmapped.

Superclasses    `<frame-event>`

Init-keywords   None.

Description      The class of events that indicate a frame has been unmapped, that is, removed from the screen. An instance of this class is distributed whenever a frame is unmapped. A frame may be unmapped by either iconifying it, or by exiting or destroying the frame completely, so that it no longer exists.

Operations     None.

Example        The following example defines a method that can inform you when an instance of a class of frame you have defined is unmapped.

```
define method handle-event
    (frame :: <my-frame>,
     event :: <frame-unmapped-event>) => ()
    notify-user
        (format-to-string("Frame %= unmapped", frame))
end method handle-event;
```

See also        `<frame-mapped-event>`, page 711

## \*global-command-table\*

### *Variable*

Summary	The command table inherited by all new command tables.
Type	<code>&lt;command-table&gt;</code>
Description	This is the command table from which all other command tables inherit by default. You should not explicitly add anything to or remove anything from this command table. DUIM can use this command to store internals or system-wide commands. You should not casually install any commands or translators into this command table.
See also	<code>&lt;command-table&gt;</code> , page 634 <code>*user-command-table*</code> , page 750

## iconify-frame

### *Generic function*

Summary	Iconifies the specified frame.
Signature	<code>iconify-frame frame =&gt; ()</code>
Arguments	<code>frame</code> An instance of type <code>&lt;frame&gt;</code> .
Values	None
Description	Iconifies <code>frame</code> . The appearance of the iconified frame depends on the behavior of the operating system in which the application is running. For instance, in Windows 95 or Windows NT 4.0, the icon is displayed in the task bar at the bottom of the screen.
Example	The following example creates and displays a simple frame, then iconifies it. You should run this code in the interactor, pressing the RETURN key at the points indicated.

```

define variable *frame* =
  make(<simple-frame>, title: "A frame",
       layout: make(<button>));RETURN

start-frame(*frame*);RETURN
iconify-frame(*frame*);RETURN

```

See also      [deiconify-frame](#), page 655

[destroy-frame](#), page 655

[exit-frame](#), page 692

[frame-icon](#), page 707

[lower-frame](#), page 727

[raise-frame](#), page 741

## layout-frame *Generic function*

Summary      Resizes the specified frame and lays out the current pane hierarchy inside it.

Signature      **layout-frame** *frame* #key **width height** => ()

Arguments      **frame**      An instance of type <**frame**>.  
**width**      An instance of type **false-or(<integer>)**.  
**height**      An instance of type **false-or(<integer>)**.

Values      None

Description      Resizes the frame and lays out the current pane hierarchy according to the layout protocol. This function is automatically invoked on a frame when it is adopted, after its pane hierarchy has been generated.

If *width* and *height* are provided, then this function resizes the frame to the specified size. It is an error to provide just *width*.

If no optional arguments are provided, this function resizes the frame to the preferred size of the top-level pane as determined by the space composition pass of the layout protocol.

In either case, after the frame is resized, the space allocation pass of the layout protocol is invoked on the top-level pane.

## **lower-frame**

### *Generic function*

**Summary**      Lowers the specified frame to the bottom of the stack of visible windows.

**Signature**      `lower-frame frame => ()`

**Arguments**      `frame`      An instance of type `<frame>`.

**Values**      None

**Description**      Lowers `frame` to the bottom of the stack of visible windows. After calling this function, `frame` will appear beneath any occluding windows that may be on the screen.

**Example**      The following example creates and displays a simple frame, then lowers it.

```
define variable *frame* =
    make(<simple-frame>, title: "A frame",
         layout: make(<button>));

start-frame(*frame*);
lower-frame(*frame*);
```

**See also**      [deiconify-frame](#), page 655  
[destroy-frame](#), page 655  
[exit-frame](#), page 692  
[iconify-frame](#), page 725  
[raise-frame](#), page 741

make	<i>G.f. method</i>
Summary	Creates an instance of a <frame>.
Signature	<pre>make (class == &lt;frame&gt;)       #key top-level command-queue layout icon            pointer-documentation command-table menu-bar tool-bar            status-bar title calling-frame top-level-sheet state            geometry resizable? properties thread event-queue            foreground background text-style palette save-under?            drop-shadow? dialog-for =&gt; simple-frame</pre>
Arguments	<p><i>class</i>      The class &lt;frame&gt;.</p> <p><i>top-level</i>    An instance of type <code>false-or(&lt;sheet&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>command-queue</i>    An instance of type <code>false-or(&lt;event-queue&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>layout</i>      An instance of type <code>false-or(&lt;sheet&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>icon</i>        An instance of type <code>false-or(&lt;image&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>pointer-documentation</i>    An instance of type <code>false-or(&lt;string&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>command-table</i>    An instance of type <code>false-or(&lt;command-table&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>menu-bar</i>     An instance of type <code>false-or(&lt;menu-bar&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>tool-bar</i>     An instance of type <code>false-or(&lt;tool-bar&gt;)</code>. Default value: <code>#f</code>.</p> <p><i>status-bar</i>    An instance of type <code>false-or(&lt;status-bar&gt;)</code>. Default value: <code>#f</code>.</p>

<i>title</i>	An instance of type <code>false-or(&lt;string&gt;)</code> . Default value: <code>#f</code> .
<i>calling-frame</i>	An instance of type <code>false-or(&lt;frame&gt;)</code> . Default value: <code>#f</code> .
<i>state</i>	An instance of type <code>one-of(#"detached", "#unmapped", "#mapped", "#iconified")</code> . Default value: <code>#"detached"</code> .
<i>geometry</i>	An instance of type <code>&lt;vector&gt;</code> . Default value: <code>vector(#f, #f, #f, #f)</code> .
<i>resizable?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#t</code> .
<i>properties</i>	An instance of type <code>&lt;stretchy-object-vector&gt;</code> . Default value: <code>make(&lt;stretchy-vector&gt;)</code> .
<i>thread</i>	An instance of type <code>false-or(&lt;thread&gt;)</code> . Default value: <code>#f</code> .
<i>event-queue</i>	An instance of type <code>false-or(&lt;event-queue&gt;)</code> . Default value: <code>#f</code> .
<i>foreground</i>	An instance of type <code>false-or(&lt;ink&gt;)</code> . Default value: <code>#f</code> .
<i>background</i>	An instance of type <code>false-or(&lt;ink&gt;)</code> . Default value: <code>#f</code> .
<i>text-style</i>	An instance of type <code>false-or(&lt;text-style&gt;)</code> . Default value: <code>#f</code> .
<i>palette</i>	An instance of type <code>false-or(&lt;palette&gt;)</code> . Default value: <code>#f</code> .
<i>save-under?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#f</code> .
<i>drop-shadow?</i>	An instance of type <code>&lt;boolean&gt;</code> . Default value: <code>#f</code> .
<i>dialog-for</i>	An instance of type <code>&lt;dialog-frame&gt;</code> .

Values	<i>simple-frame</i>	An instance of type < <b>frame</b> >.
Description		<p>Creates and returns an instance of &lt;<b>frame</b>&gt; or one of its sub-classes.</p> <p>The <i>top-level</i> argument specifies the top-level command-loop in which the frame runs.</p> <p>The <i>command-queue</i> argument specifies a command-queue for the frame.</p> <p>The <i>layout</i> argument specifies a layout for the frame.</p> <p>The <i>icon</i> argument specifies an icon that will be used when the frame is iconized. In all current versions of Windows, this icon is also visible in the left hand corner of the title bar of the frame when it is not iconized.</p> <p>The <i>pointer-documentation</i> argument specifies pointer-documentation for the frame.</p> <p>The <i>command-table</i> argument specifies a command table for the frame.</p> <p>The <i>menu-bar</i> argument specifies a menu bar for the frame.</p> <p>The <i>tool-bar</i> argument specifies a tool bar for the frame.</p> <p>The <i>status-bar</i> argument specifies a status bar for the frame.</p> <p>The <i>title</i> argument specifies a title for the frame.</p> <p>The <i>calling-frame</i> argument specifies a calling frame for the frame.</p> <p>The <i>state</i> argument specifies a frame-state. The frame can be mapped or unmapped (that is, visible on the screen, or not), iconified, or detached.</p> <p>The <i>geometry</i> argument specifies a for the frame. The four components of this keyword represent the x and y position of the frame, and the width and height of the frame, respectively.</p>

The *resizable?* argument specifies whether or not the frame is resizable.

The *properties* argument specifies properties for the frame.

The *thread* argument specifies the thread in which the frame will run. See the *Library Reference: Core Features* manual for full details about how threads are handled.

The *event-queue* specifies an event-queue for the frame.

The arguments *foreground* and *background* specify a foreground color for the frame. In addition, *text-style* specifies a text style for the frame, and *palette* specifies a color palette for the frame.

See also      [`<frame>`](#), page 694

## **make-menu-from-command-table-menu** *Generic function*

**Summary**      Returns a menu from the menu definition in the specified command table.

**Signature**      `make-menu-from-command-table-menu  
      command-table-menu-items frame framem  
      #:key command-table label mnemonic item-callback  
=> menu`

**Arguments**      *command-table-menu-items*  
                         An instance of type [`<sequence>`](#).  
*frame*              An instance of type [`<frame>`](#).  
*framem*             An instance of type [`<frame-manager>`](#).  
*command-table*      An instance of type [`<command-table>`](#).  
*label*               An instance of type [`<label>`](#).  
*mnemonic*           An instance of type `false-or(<gesture>)`.  
*item-callback*      An instance of type [`<function>`](#).

Values	<i>menu</i>	An instance of type < <code>menu</code> >.
Description		<p>Returns a menu from the menu definition in the specified command table. This function is used by <code>make-menus-from-command-table</code> to individually create each menu defined in the command table. The function <code>make-menus-from-command-table</code> then puts each of the menus created together in the appropriate way.</p> <p>The <i>command-table-menu-items</i> argument defines the items that are to be placed in the menu. It is a sequence of instances of &lt;<code>command-table-menu-item</code>&gt;.</p> <p>The <i>frame</i> and <i>framem</i> arguments define the frame and the frame manager in which the menu created is to be placed.</p> <p>The <i>command-table</i> argument specifies the command table in which the definition of the menu created can be found.</p> <p>The <i>label</i> argument defines a label for the menu created.</p> <p>The <i>mnemonic</i> argument defines a keyboard mnemonic for the menu created.</p>
See also		<code>make-menus-from-command-table</code> , page 732

	<b>make-menus-from-command-table</b>	<i>Generic function</i>
Summary		Returns a set of menus from the menu definitions in the specified command table.
Signature		<code>make-menus-from-command-table command-table frame framem #key label =&gt; menus</code>
Arguments	<p><i>command-table</i></p> <p><i>frame</i></p> <p><i>framem</i></p> <p><i>label</i></p>	<p>An instance of type &lt;<code>command-table</code>&gt;.</p> <p>An instance of type &lt;<code>frame</code>&gt;.</p> <p>An instance of type &lt;<code>frame-manager</code>&gt;.</p> <p>An instance of type &lt;<code>label</code>&gt;.</p>

Values	<i>menus</i>	An instance of type <code>limited(&lt;sequence&gt;, of: &lt;menu&gt;)</code> .
Description	Returns a set of menus from the menu definitions in <i>command-table</i> .	The <i>frame</i> and <i>framem</i> arguments specify the frame and frame manager in which the menus are to be placed.
		The <i>label</i> argument lets you specify a label for the set of menus.
See also		<code>make-menu-from-command-table-menu</code> , page 731

## menu-item-accelerator *Generic function*

Summary	Returns the accelerator for the specified command table menu item.	
Signature	<code>menu-item-accelerator menu-item =&gt; accelerator</code>	
Arguments	<i>menu-item</i>	An instance of type <code>&lt;command-table-menu-item&gt;</code> .
Values	<i>accelerator</i>	An instance of type <code>&lt;gesture&gt;</code> .
Description	Returns the keyboard accelerator for <i>menu-item</i> . Note that <i>menu-item</i> must be defined in a command table.	
See also		<code>menu-item-mnemonic</code> , page 733

## menu-item-mnemonic *Generic function*

Summary	Returns the mnemonic for the specified menu item.
Signature	<code>menu-item-mnemonic menu-item =&gt; mnemonic</code>

Arguments	<i>menu-item</i>	An instance of type <command-table-menu-item>.
Values	<i>mnemonic</i>	An instance of type <b>false-or(&lt;gesture&gt;)</b> .
Description		Returns the keyboard mnemonic for <i>menu-item</i> .
See also		<a href="#">menu-item-accelerator</a> , page 733

## **menu-item-name** *Generic function*

Summary		Returns the name of the specified menu item.
Signature		<b>menu-item-name</b> <i>menu-item</i> => <i>name</i>
Arguments	<i>menu-item</i>	An instance of type <command-table-menu-item>.
Values	<i>name</i>	An instance of type <string>.
Description		Returns the name of <i>menu-item</i> .
See also		<a href="#">menu-item-options</a> , page 734 <a href="#">menu-item-type</a> , page 735 <a href="#">menu-item-value</a> , page 735

## **menu-item-options** *Generic function*

Summary		Returns the options for the specified menu item.
Signature		<b>menu-item-options</b> <i>menu-item</i> => <i>options</i>
Arguments	<i>menu-item</i>	An instance of type <command-table-menu-item>.

Values	<i>options</i>	An instance of type <object>.
Description	Returns the options for <i>menu-item</i> .	
See also	<a href="#">menu-item-name</a> , page 734 <a href="#">menu-item-type</a> , page 735 <a href="#">menu-item-value</a> , page 735	

## menu-item-type *Generic function*

Summary	Returns the type of the specified menu item.	
Signature	<code>menu-item-type menu-item =&gt; type</code>	
Arguments	<i>menu-item</i>	An instance of type <command-table-menu-item>.
Values	<i>type</i>	An instance of type <object>.
Description	Returns the type of <i>menu-item</i> .	
See also	<a href="#">menu-item-name</a> , page 734 <a href="#">menu-item-options</a> , page 734 <a href="#">menu-item-value</a> , page 735	

## menu-item-value *Generic function*

Summary	Returns the value of the specified menu item.	
Signature	<code>menu-item-value menu-item =&gt; value</code>	
Arguments	<i>menu-item</i>	An instance of type <command-table-menu-item>.

Values	<i>value</i>	An instance of type <object>.
Description		Returns the value of <i>menu-item</i> .
See also		<code>menu-item-name</code> , page 734 <code>menu-item-options</code> , page 734 <code>menu-item-type</code> , page 735

## **move-to-next-page** *Generic function*

Summary	Moves to the next page of the specified multi-page dialog.
Signature	<code>move-to-next-page wizard =&gt; ()</code>
Arguments	<i>wizard</i>
Values	An instance of type <wizard-frame>.
Description	Moves to the next page in sequence of <i>wizard</i> . This is the default callback for the Next button in a wizard frame.
See also	<code>dialog-next-callback</code> , page 678 <code>&lt;wizard-frame&gt;</code> , page 751

## **move-to-previous-page** *Generic function*

Summary	Moves to the previous page of the specified multi-page dialog.
Signature	<code>move-to-previous-page wizard =&gt; ()</code>
Arguments	<i>wizard</i>

Values	None.
Description	Moves to the previous page in sequence of <i>wizard</i> . This is the default callback for the Back button in a wizard frame.
See also	<code>dialog-back-callback</code> , page 659 <code>&lt;wizard-frame&gt;</code> , page 751

## **note-progress** *Generic function*

Summary	Note the progress of an event in the specified progress note.										
Signature	<code>note-progress numerator denominator</code> <code>#key note label pointer-cursor =&gt; ()</code>										
Arguments	<table> <tr> <td><i>numerator</i></td><td>An instance of type <code>&lt;integer&gt;</code>.</td></tr> <tr> <td><i>denominator</i></td><td>An instance of type <code>&lt;integer&gt;</code>.</td></tr> <tr> <td><i>note</i></td><td>An instance of type <code>&lt;progress-note&gt;</code>. Default value: <code>*progress-note*</code>.</td></tr> <tr> <td><i>label</i></td><td>An instance of type <code>&lt;label&gt;</code>.</td></tr> <tr> <td><i>pointer-cursor</i></td><td>An instance of type <code>&lt;pointer&gt;</code>.</td></tr> </table>	<i>numerator</i>	An instance of type <code>&lt;integer&gt;</code> .	<i>denominator</i>	An instance of type <code>&lt;integer&gt;</code> .	<i>note</i>	An instance of type <code>&lt;progress-note&gt;</code> . Default value: <code>*progress-note*</code> .	<i>label</i>	An instance of type <code>&lt;label&gt;</code> .	<i>pointer-cursor</i>	An instance of type <code>&lt;pointer&gt;</code> .
<i>numerator</i>	An instance of type <code>&lt;integer&gt;</code> .										
<i>denominator</i>	An instance of type <code>&lt;integer&gt;</code> .										
<i>note</i>	An instance of type <code>&lt;progress-note&gt;</code> . Default value: <code>*progress-note*</code> .										
<i>label</i>	An instance of type <code>&lt;label&gt;</code> .										
<i>pointer-cursor</i>	An instance of type <code>&lt;pointer&gt;</code> .										
Values	None										
Description	<p>Note the progress of an event in <i>note</i>.</p> <p>If a <i>numerator</i> and <i>denominator</i> are supplied, then the progress is displayed in terms of those figures. For example, if <i>numerator</i> is 1, and <i>denominator</i> is 10, then the progress is displayed in tenths.</p> <p>If supplied, <i>pointer-cursor</i> is used as a cursor when the mouse pointer is placed over the owner frame.</p>										
See also	<code>noting-progress</code> , page 738										

**\*progress-note\***, page 738**noting-progress***Statement macro*

Summary	Performs a body of code, noting its progress.						
Macro call	<code>noting-progress ( {sheet} , {label} ) {body} end</code>						
Arguments	<table> <tr> <td><i>sheet</i></td><td>A Dylan expression<sub>bnf</sub>.</td></tr> <tr> <td><i>label</i></td><td>A Dylan expression<sub>bnf</sub>.</td></tr> <tr> <td><i>body</i></td><td>A Dylan body<sub>bnf</sub>.</td></tr> </table>	<i>sheet</i>	A Dylan expression <sub>bnf</sub> .	<i>label</i>	A Dylan expression <sub>bnf</sub> .	<i>body</i>	A Dylan body <sub>bnf</sub> .
<i>sheet</i>	A Dylan expression <sub>bnf</sub> .						
<i>label</i>	A Dylan expression <sub>bnf</sub> .						
<i>body</i>	A Dylan body <sub>bnf</sub> .						
Values	None.						
Description	<p>Performs a body of code, noting its progress, for the specified sheet.</p> <p>The sheet argument is an expression that evaluates to an instance of <code>&lt;sheet&gt;</code>. The label argument is an expression that evaluates to an instance of <code>&lt;string&gt;</code>.</p>						
See also	<code>note-progress</code> , page 737						

**\*progress-note\****Fluid variable*

Summary	Specifies a default progress note that can be used.
Type	<code>&lt;object&gt;</code>
Initial value	<code>#f</code>
Description	This variable is used to supply a default progress note to use if no progress note is explicitly specified.
See also	<code>note-progress</code> , page 737

**<property-frame>***Open instantiable class*

**Summary**      The class of property frames.

**Superclasses**    <**dialog-frame**>

**Init-keywords**    **pages:**      An instance of type `false-or(limited(<sequence>, of: <page>))`.  
Default value: #f.

**apply-callback:**

An instance of type `false-or(<function>)`.  
Default value: #f.

**apply-button:** An instance of type `false-or(<button>)`.  
Default value: #f.

**Note:** The following two useful init-keywords are inherited from <**dialog-frame**>:

**pages:**      An instance of type `false-or(<sequence>)`.  
Default value: #f.

**page-changed-callback:**

An instance of type `false-or(<function>)`.  
Default value: #f.

**Description**      The class of property frames. These are dialogs that can contain property sheets of some description. This is the class of dialogs with several pages, each presented as a label in a tab control.

The **pages:** init-keyword defines the pages available for the property frame.

The apply callback and button define an additional Apply button available in property frames. The Apply button applies any changes made in the current page of the dialog, but does not dismiss the dialog from the screen. By default, there is no Apply button defined.



**Figure 9.3** A property frame

The page-changed callback lets you specified a callback that should be invoked if the current page in the property frame is changed by clicking on a different page tab.

**Operations**      The following operations are exported from the **DUIM-Frames** module.

```
dialog-apply-button dialog-apply-button-setter  
dialog-apply-callback dialog-current-page  
dialog-current-page-setter  
dialog-page-changed-callback dialog-page-changed-  
callback-setter dialog-page-complete? dialog-page-  
complete?-setter dialog-pages dialog-pages-setter
```

**See also**      [dialog-apply-button](#), page 656

[dialog-apply-callback](#), page 657

[<dialog-frame>](#), page 670

[<property-page>](#), page 740

[<wizard-frame>](#), page 751

## <property-page>

*Open instantiable class*

**Summary**      The class of property pages.

**Superclasses**    [<page>](#)

Init-keywords	None.
Description	The class of property pages. These are pages that can be displayed in an instance of < <b>property-frame</b> >.



**Figure 9.4** A property page

Internally, this class maps into the Windows property page control.

Operations	None.
See also	<a href="#">&lt;page&gt;</a> , page 550 <a href="#">&lt;property-frame&gt;</a> , page 739 <a href="#">&lt;property-page&gt;</a> , page 740 <a href="#">&lt;tab-control-page&gt;</a> , page 582 <a href="#">&lt;wizard-page&gt;</a> , page 754

## raise-frame *Generic function*

Summary	Raises the specified frame to the top of the stack of visible windows.	
Signature	<code>raise-frame <i>frame</i> =&gt; ()</code>	
Arguments	<i>frame</i>	An instance of type < <b>frame</b> >.
Values	None	

Description      Raises *frame* to the top of the stack of visible windows. After calling this function, *frame* will appear above any occluding windows that may be on the screen.

Example      The following example creates and displays a simple frame, then lowers and raises it. You should run this code in the interactor, pressing the RETURN key at the points indicated.

```
define variable *frame* =
    make(<simple-frame>, title: "A frame",
        layout: make(<button>));RETURN

start-frame(*frame*);RETURN
lower-frame(*frame*);RETURN
raise-frame(*frame*);RETURN
```

See also      [deiconify-frame](#), page 655  
[destroy-frame](#), page 655  
[exit-frame](#), page 692  
[iconify-frame](#), page 725  
[lower-frame](#), page 727

## redo-command

*Generic function*

Summary      Performs the last performed command again.

Signature      `redo-command command frame => #rest values`

Arguments      *command*      An instance of type <command>.  
*frame*          An instance of type <frame>.

Values          *values*        Instances of type <object>.

Description      Performs *command* again. The command is the command that was last executed using [execute-command](#).

Note that the command described by *command* must be undoable.

You can both specialize this function and call it directly in your code.

See also [execute-command](#), page 690

## **remove-command** *Generic function*

Summary Removes a command from the specified command table.

Signature `remove-command command-table command => ()`

Arguments *command-table* An instance of type `<command-table>`.

*command* An instance of type `<command>`.

Values None

Description Removes *command* from *command-table*.

See also [add-command](#), page 621

## **remove-command-table** *Function*

Summary Removes the specified command table.

Signature `remove-command-table command-table => ()`

Arguments *command-table* An instance of type `<command-table>`.

Values None

Description Removes *command-table*.

		<i>Generic function</i>
	<b>remove-command-table-menu-item</b>	
Summary	Removes a menu item from the specified command table.	
Signature	<code>remove-command-table-menu-item <i>command-table</i> <i>string</i> =&gt; ()</code>	
Arguments	<i>command-table</i> An instance of type <command-table>. <i>string</i> An instance of type <string>.	
Values	None	
Description	Removes the menu item identified by <i>string</i> from <i>command-table</i> .	
See also	<a href="#">add-command-table-menu-item</a> , page 623	
	<b>set-frame-position</b>	<i>Generic function</i>
Summary	Sets the position of the specified frame.	
Signature	<code>set-frame-position <i>frame</i> <i>x</i> <i>y</i> =&gt; ()</code>	
Arguments	<i>frame</i> An instance of type <frame>. <i>x</i> An instance of type <integer>. <i>y</i> An instance of type <integer>.	
Values	None	
Description	Sets the position of <i>frame</i> . The coordinates <i>x</i> and <i>y</i> are measured from the top left of the screen, measured in pixels.	
See also	<a href="#">frame-position</a> , page 716 <a href="#">set-frame-size</a> , page 745	

## set-frame-size

*Generic function*

Summary Sets the size of the specified frame.

Signature `set-frame-size frame width height => ()`

Arguments `frame` An instance of type `<frame>`.  
`width` An instance of type `<integer>`.  
`height` An instance of type `<integer>`.

Values None

Description Sets the size of `frame`.

Example The following example creates and displays a simple frame, then resizes it. You should run this code in the interactor, pressing the RETURN key at the points indicated.

```
define variable *frame* =
    make(<simple-frame>, title: "A frame",
        layout: make(<button>, label: "Button"));RETURN
set-frame-size(*frame*, 100, 500);RETURN
```

See also [frame-size](#), page 717

[set-frame-position](#), page 744

## <simple-command>

*Open abstract instantiable class*

Summary The class of simple commands.

Superclasses `<object>`

Init-keywords `function:` An instance of type `<function>`. Required.

`arguments:` An instance of type `<sequence>`. Default value `#[]`.

Description	The class of simple commands. A simple command has an associated function and some arguments. Simple commands are not undoable.  The first argument to the function is always the frame.
Operations	None.
See also	<command>, page 630

## <simple-frame>

*Open abstract instantiable class*

Summary	The class of simple frames.
Superclasses	<frame>
Init-keywords	<b>command-queue:</b> An instance of type <code>false-or(&lt;event-queue&gt;)</code> . Default value: <code>make(&lt;event-queue&gt;)</code> .
	<b>layout:</b> An instance of type <code>false-or(&lt;sheet&gt;)</code> . Default value: <code>#f</code> .
	<b>command-table:</b> An instance of type <code>false-or(&lt;command-table&gt;)</code> . Default value: <code>#f</code> .
	<b>menu-bar:</b> An instance of type <code>false-or(&lt;menu-bar&gt;)</code> . Default value: <code>#f</code> .
	<b>tool-bar:</b> An instance of type <code>false-or(&lt;tool-bar&gt;)</code> . Default value: <code>#f</code> .
	<b>status-bar:</b> An instance of type <code>false-or(&lt;status-bar&gt;)</code> . Default value: <code>#f</code> .
Description	The class of simple frames.  The <code>command-queue:</code> init-keyword specifies a command-queue for the frame.

The `layout`: init-keyword specifies a layout for the frame.

The `command-table`: init-keyword specifies a command table for the frame.

The `menu-bar`: init-keyword specifies a menu bar for the frame.

The `tool-bar`: init-keyword specifies a tool bar for the frame.

The `status-bar`: init-keyword specifies a status bar for the frame.

**Operations** The following operations are exported from the `DUIM-Frames` module.

```
frame-command-table frame-command-table-setter frame-
layout frame-layout-setter frame-menu-bar frame-menu-
bar-setter frame-status-bar frame-status-bar-setter
frame-status-message frame-status-message-setter
frame-tool-bar frame-tool-bar-setter frame-top-level
start-frame
```

## <simple-undoable-command> *Open abstract instantiable class*

**Summary** The class of simple commands that can contain an undo action.

**Superclasses** `<object>`

**Init-keywords** `undo-command`: An instance of type `<command>`.

**Description** The class of simple commands that can contain an undo action. A simple undoable command is like a simple command, except that it points to a command that can undo it, represented by the `undo-command`: init-keyword.

**Operations** None.

See also      `<simple-command>`, page 745

## **start-dialog** *Generic function*

Summary	Displays a DUIM frame as a dialog box.	
Signature	<code>start-dialog dialog =&gt; #rest values</code>	
Arguments	<i>dialog</i>	An instance of type <code>&lt;dialog-frame&gt;</code> .
Values	<i>values</i>	Instances of type <code>&lt;object&gt;</code> .
Description	<p>Displays a DUIM frame as a dialog box.</p> <p>The function <code>start-dialog</code> dynamically binds an <code>&lt;abort&gt;</code> restart around the event loop for the dialog that is started. The restart allows the event loop to be re-entered, and enables any callbacks run from the dialog to signal an <code>&lt;abort&gt;</code> (via the <code>abort</code> function, for instance), in order to terminate execution of the current callback and return to event processing. This facility is useful for implementing operations that cancel gestures and for debugging DUIM applications from Dylan debuggers.</p>	

See also      `cancel-dialog`, page 629  
                 `<dialog-frame>`, page 670  
                 `exit-dialog`, page 691  
                 `start-frame`, page 748

## **start-frame** *Generic function*

Summary	Starts the specified frame.
Signature	<code>start-frame frame #key owner mode =&gt; status-code</code>

Arguments	<i>frame</i>	An instance of type <frame>.
	<i>owner</i>	An instance of type <b>false-or(&lt;frame&gt;)</b> . Default value: #f.
	<i>mode</i>	An instance of type <b>one-of("modal", "#\"modeless\"", "#\"system-modal\"")</b> . Default value: #f.
Values	<i>status-code</i>	An instance of type <integer>.
Description		<p>Starts <i>frame</i>, optionally setting the <i>owner</i> of the frame and the <i>mode</i> in which it will run.</p> <p>The function <b>start-frame</b> dynamically binds an &lt;abort&gt; restart around the event loop for the frame that is started. The restart allows the event loop to be re-entered, and enables any callbacks run from the frame to signal an &lt;abort&gt; (via the <b>abort</b> function, for instance), in order to terminate execution of the current callback and return to event processing. This facility is useful for implementing operations that cancel gestures and for debugging DUIM applications from Dylan debuggers.</p>
Example		<p>The following example creates a simple frame, then displays it. You should run this code in the interactor, pressing the RETURN key at the points indicated.</p> <pre>define variable *frame* =   make(&lt;simple-frame&gt;, title: "A frame",        layout: make(&lt;button&gt;));RETURN  start-frame(*frame*);RETURN</pre>
See also		<p><b>exit-frame</b>, page 692</p> <p><b>frame-mapped?-setter</b>, page 712</p> <p><b>start-dialog</b>, page 748</p>

		<i>Generic function</i>
Summary	Calls the undo command for the specified command.	
Signature		<code>undo-command <i>command frame</i> =&gt; #rest <i>values</i></code>
Arguments	<i>command</i>	An instance of type <code>&lt;command&gt;</code> .
	<i>frame</i>	An instance of type <code>&lt;frame&gt;</code> .
Values	<i>values</i>	Instances of type <code>&lt;object&gt;</code> .
Description	Calls the undo command for <i>command</i> , undoing the effects of calling <i>command</i> . Note that <i>command</i> must be undoable.  You can call this command directly in your own code, as well as specialize it.	
See also		<code>command-undoable?</code> , page 640

		<i>Variable</i>
Summary	A user-defined command table that can be inherited by other command tables.	
Type		<code>&lt;command-table&gt;</code>
Description	This is a command table that can be used by the programmer for any purpose. DUIM does not use it for anything, and its contents are completely undefined.  If desired, all new command tables can inherit the command table specified by this variable.	
See also		<code>&lt;command-table&gt;</code> , page 634 <code>*global-command-table*</code> , page 725

## <wizard-frame> *Open instantiable class*

Summary	The class of wizard frames.
Superclasses	<dialog-frame>
Init-keywords	<p><b>page:</b> An instance of type &lt;page&gt;.</p> <p><b>pages:</b> An instance of type <code>false-or(limited(&lt;sequence&gt;, of: &lt;page&gt;))</code>. Default value: #f.</p> <p><b>apply-callback:</b> An instance of type <code>false-or(&lt;function&gt;)</code>. Default value: #f.</p> <p><b>apply-button:</b> An instance of type <code>false-or(&lt;button&gt;)</code>. Default value: #f.</p>
	Note that the following two useful init-keywords are inherited from <dialog-frame>:
	<p><b>pages:</b> An instance of type <code>false-or(&lt;sequence&gt;)</code>. Default value: #f.</p> <p><b>page-changed-callback:</b> An instance of type <code>false-or(&lt;function&gt;)</code>. Default value: #f.</p>
Description	<p>The class of wizard frames. These are frames that are used to create wizards (series of connected dialogs) that are used to guide the user through a structured task, such as installing an application.</p> <p>A wizard frame is a multi-page dialog, in which the user specifies requested information before proceeding to the next page in the sequence. At the end of the sequence, the user exits the dialog to send the relevant information back to the controlling application.</p>



**Figure 9.5** A wizard frame

When a wizard frame is created, each page in the frame automatically has a Next and Back button to let the user navigate forward and backward through the sequence of pages.

In addition, if `apply-button:` is specified, an Apply button is displayed in the frame. By default, clicking on this button lets the user apply the changes made so far without dismissing the frame from the screen. If specified, the `apply-callback:` function is invoked when the Apply button is clicked.

The layout of a wizard frame is controlled using a `<stack-layout>`.

**Operations** The following operations are exported from the `DUIM-Frames` module.

```
compute-next-page compute-previous-page dialog-back-
button dialog-back-button-setter dialog-back-call-
back dialog-current-page dialog-current-page-setter
dialog-next-button dialog-next-button-setter dialog-
next-callback dialog-next-enabled?
dialog-next-enabled?-setter dialog-next-page dialog-
next-page-setter dialog-page-changed-callback dialog-
page-changed-callback-setter dialog-page-complete?
dialog-page-complete?-setter dialog-pages dialog-
pages-setter dialog-previous-page dialog-previous-
page-setter move-to-next-page move-to-previous-page
```

**Example**

```

define frame <my-wizard> (<wizard-frame>
  pane name-pane (frame)
    make(<text-field>);
  pane organization-pane (frame)
    make(<text-field>);
  pane job-description-pane (frame)
    make(<text-field>);
  pane years-employed-pane (frame)
    make(<text-field>, value-type: <integer>);
  pane first-page-layout (frame)
    make(<table-layout>,
      columns: 2,
      x-alignment: #(#"right", #"left"),
      children: vector(make(<label>,
        label: "Name:"), frame.name-pane,
        make(<label>,
          label: "Organization:"), frame.organization-pane));
  pane second-page-layout (frame)
    make(<table-layout>,
      columns: 2,
      x-alignment: #(#"right", #"left"),
      children: vector
        (make(<label>,
          label: "Job Description:"), frame.job-description-pane,
        make(<label>,
          label: "Years Employed:"), frame.years-employed-pane));
  pane first-page (frame)
    make(<wizard-page>,
      child: frame.first-page-layout);
  pane second-page (frame)
    make(<wizard-page>,
      child: frame.second-page-layout);
pages (frame)
  vector(frame.first-page, frame.second-page);
keyword title: = "My Wizard";
end frame <my-wizard>;

```

**See also**

<[dialog-frame
<\[property-frame\]\(#\)>, page 739  
<\[wizard-page\]\(#\)>, page 754](#)

## <wizard-page>

*Open instantiable class*

Summary      The class of wizard pages.

Superclasses    <[page](#)>

Init-keywords   None.

Description      The class of wizard pages. These are pages that can be displayed in an instance of <[wizard-frame](#)>, and are used for a single dialog in the structured task that the wizard guides the user through.



**Figure 9.6** A wizard page

Operations     None.

See also       <[page](#)>, page 550

<[property-page](#)>, page 740

<[tab-control-page](#)>, page 582

<[wizard-frame](#)>, page 751

---

---

# Index

## Symbols

`<clipboard>` class 208  
= method 10, 104, 183, 621

## A

abort-path generic function 351, 354  
accelerator: init-keyword  
  for `<button>` 470  
  for `<command-table-menu-item>` 638  
accepts-focus?: init-keyword  
  for `<sheet>` 300  
`<action-gadget>` class 466  
activate-callback: init-keyword  
  for `<action-gadget>` 466  
activate-gadget generic function 466  
add-child generic function 184  
add-clipboard-data generic function 184  
add-clipboard-data-as generic  
  function 185  
add-colors generic function 105  
add-column generic function 467  
add-command generic function 621  
add-command-line-name generic  
  function 623  
add-command-table-menu-item generic  
  function 623  
add-item generic function 467  
add-node generic function 468  
add-presentation-translator generic  
  function 627  
alignment: init-keyword  
  for `<table-column>` 585  
allocate-space generic function 399  
`$altgr-key` constant 186  
`$alt-key` constant 185

`<application-exited-event>` class 627  
apply-button: init-keyword  
  for `<property-frame>` 739  
  for `<wizard-frame>` 751  
apply-callback: init-keyword  
  for `<property-frame>` 739  
  for `<wizard-frame>` 751  
apply-in-frame generic function 628  
`arc-to` generic function 353, 354  
`<area>` class 10  
area? generic function 11  
arguments: init-keyword  
  for `<command>` 631  
  for `<simple-command>` 745  
array: init-keyword  
  for `<stencil>` 161  
auto-scroll?: init-keyword  
  for `<text-field>` 594

## B

`$background` constant 105  
background: init-keyword  
  for `<brush>` 111, 140  
  for `<frame>` 696  
  for `<sheet>` 300  
`<basic-composite-pane>` class 400  
`<basic-user-pane>` class 400  
beep generic function 186  
`$black` constant 105  
`$blue` constant 106  
blue: init-keyword  
  for `<color>` 117  
`$boole-1` constant 107  
`$boole-2` constant 107  
`$boole-and` constant 108

**\$boole-andc1** constant 110  
**\$boole-andc2** constant 110  
**\$boole-c1** constant 107  
**\$boole-c2** constant 108  
**\$boole-clr** constant 106  
**\$boole-eqv** constant 109  
**\$boole-ior** constant 108  
**\$boole-nand** constant 109  
**\$boole-nor** constant 109  
**\$boole-orc1** constant 110  
**\$boole-orc2** constant 111  
**\$boole-set** constant 106  
**\$boole-xor** constant 108  
**<border>** class 469  
**border:** init-keyword  
  for **<column-layout>** 401  
  for **<row-layout>** 423  
  for **<stack-layout>** 433  
  for **<table-layout>** 436  
**borders:** init-keyword  
  for **<combo-box>** 481  
  for **<list-control>** 531  
  for **<option-box>** 549  
  for **<slider>** 567  
  for **<spin-box>** 570  
  for **<table-control>** 586  
**bottom:** init-keyword  
  for **<bounding-box>** 12  
**boundary-event-kind** generic  
  function 186  
**<bounding-box>** class 11  
**bounding-box** generic function 13  
**bounding-box?** generic function 12  
**box-bottom** function 14  
**box-edges** function 14  
**box-height** function 15  
**box-left** function 16  
**box-position** function 16  
**box-right** function 17  
**box-size** function 18  
**box-top** function 18  
**box-width** function 19  
**\$bricks-stipple** constant 111  
**<brush>** class 111  
**brush?** generic function 112  
**brush-background** generic function 113  
**brush-fill-rule** generic function 113  
**brush-fill-style** generic function 114  
**brush-foreground** generic function 114  
**brush-mode** generic function 114  
**brush-stipple** generic function 115  
**brush-stretch-mode** generic  
                        function 115  
**brush-tile** generic function 116  
**brush-ts-x** generic function 116  
**brush-ts-y** generic function 117  
**<button>** class 470  
**button:** init-keyword  
  for **<gesture>** 247  
  for **<pointer-button-event>** 280  
  for **<pointer-drag-event>** 283  
  for **<pointer-gesture>** 285  
**<button-box>** class 473  
**button-index** inline function 187  
**button-index-name** function 187  
**<button-press-event>** class 188  
**<button-release-event>** class 189

## C

**callback:** init-keyword  
  for **<table-column>** 585  
**callbacks** 585  
  activate callback 466  
  apply callback 739, 751, 752  
  cancel callback 670, 672  
  exit callback 670, 672  
  help callback 670, 673  
  key-press callback 479, 480, 531, 532,  
    578, 587, 588, 598, 600  
  page-changed callback 671, 673, 739,  
    740, 751  
  popup-menu callback 531, 532, 587, 598,  
    600  
  update callback 541, 542, 543, 544, 597  
  value-changed callback 561, 562, 609  
  value-changing callback 561, 568, 595,  
    596  
**call-in-frame** generic function 628  
**calling-frame:** init-keyword  
  for **<frame>** 695  
**cancel-button:** init-keyword  
  for **<dialog-frame>** 670  
**cancel-callback:** init-keyword  
  for **<dialog-frame>** 670  
**cancel-dialog** generic function 629  
**cap-shape:** init-keyword  
  for **<pen>** 153  
**\$capslock-key** constant 189  
**<caret>** class 189  
**caret:** init-keyword  
  for **<sheet>** 300  
**caret-position** generic function 190  
**caret-sheet** generic function 191  
**caret-size** generic function 191

**caret-visible?** generic function 192  
**caret-visible?-setter** generic function 192  
**case:** init-keyword  
   for <text-field> 594  
**cell-space-requirement:** init-keyword  
   for <grid-layout> 411  
**center-point:** init-keyword  
   for <ellipse> 70  
   for <elliptical-arc> 75  
**center-x:** init-keyword  
   for <ellipse> 70  
   for <elliptical-arc> 75  
**center-y:** init-keyword  
   for <ellipse> 70  
   for <elliptical-arc> 75  
**character:** init-keyword  
   for <keyboard-event> 255  
**<check-box>** class 475  
**<check-button>** class 476  
**<check-menu-box>** class 477  
**<check-menu-button>** class 478  
**child:** init-keyword  
   for <button-box> 474  
   for <sheet> 299  
   for <spacing> 569  
**child-containing-position** generic function 193  
**child-nodes:** init-keyword  
   for <tree-node> 607  
**children:** init-keyword  
   for <sheet> 299  
**children-generator:** init-keyword  
   for <tree-control> 598  
**children-overlapping-region** generic function 193  
**children-predicate:** init-keyword  
   for <tree-control> 598  
**choose-color** generic function 194  
**choose-directory** generic function 196  
**choose-file** generic function 198  
**choose-from-dialog** generic function 202  
**choose-from-menu** generic function 204  
**classes**  
   <action-gadget> 466  
   <application-exited-event> 627  
   <area> 10  
   <basic-composite-pane> 400  
   <basic-user-pane> 400  
   <border> 469  
   <bounding-box> 11  
   <brush> 111  
   <button> 470  
   <button-box> 473  
   <button-press-event> 188  
   <button-release-event> 189  
   <caret> 189  
   <check-box> 475  
   <check-button> 476  
   <check-menu-box> 477  
   <check-menu-button> 478  
   <clipboard> 208  
   <collection-gadget> 478  
   <color> 117  
   <color-not-found> 121  
   <column-layout> 401  
   <combo-box> 481  
   <command> 630  
   <command-table> 634  
   <command-table-menu-item> 637  
   <device-event> 214  
   <device-font> 129  
   <dialog-cancel> 660  
   <dialog-frame> 670  
   <display> 215  
   <double-click-event> 226  
   <drawing-pane> 409  
   <ellipse> 70  
   <elliptical-arc> 75  
   <event> 230  
   <fixed-layout> 410  
   <focus-gadget> 485  
   <frame> 694  
   <frame-created-event> 702  
   <frame-destroyed-event> 702  
   <frame-event> 243  
   <frame-exited-event> 704  
   <frame-exit-event> 705  
   <frame-focus-event> 706  
   <frame-manager> 243  
   <frame-mapped-event> 711  
   <frame-unmapped-event> 724  
   <gadget> 485  
   <gesture> 247  
   <grid-layout> 411  
   <group-box> 526  
   <image> 136  
   <ink> 138  
   <keyboard-event> 254  
   <keyboard-gesture> 255  
   <key-press-event> 256  
   <key-release-event> 256  
   <label> 528  
   <layout> 413

```

<leaf-pane> 417
<line> 77
<list-box> 529
<list-control> 531
<list-item> 537
<list-item-pane> 537
<medium> 259
<menu> 541
<menu-bar> 542
<menu-box> 543
<menu-button> 544
<multiple-child-composite-
    pane> 419
<null-pane> 420
<option-box> 549
<page> 550
<palette> 150
<palette-full> 151
<password-field> 550
<path> 40
<pattern> 152
<pen> 153
<pinboard-layout> 421
<pixmap> 390
<pixmap-medium> 391
<point> 41
<pointer> 278
<pointer-boundary-event> 279
<pointer-button-event> 280
<pointer-drag-event> 283
<pointer-enter-event> 283
<pointer-event> 284
<pointer-exit-event> 284
<pointer-gesture> 285
<pointer-motion-event> 285
<polygon> 86
<polyline> 89
<port> 287
<port-terminated-event> 291
<progress-bar> 551
<progress-note> 738
<property-frame> 739
<property-page> 740
<push-box> 552
<push-button> 553
<push-menu-box> 554
<push-menu-button> 555
<radio-box> 556
<radio-button> 557
<radio-menu-box> 557
<radio-menu-button> 558
<rectangle> 92
<reflection-underspecified> 45
<region> 45
<region-set> 50
<rich-text-editor> 561
<row-layout> 423
<scroll-bar> 561
<scroller> 563
<separator> 564
<sheet> 299
<sheet-event> 307
<simple-command> 745
<simple-frame> 746
<simple-pane> 426
<simple-status-bar> 567
<simple-undoable-command> 747
<single-child-composite-pane> 427
<singular-transform> 56
<slider> 567
<space-requirement> 427
<spacing> 569
<spin-box> 570
<splitter> 571, 573
<stack-layout> 433
<status-bar> 573
<stencil> 161
<tab-control> 577
<tab-control-page> 582
<table-column> 585
<table-control> 586
<table-item> 592
<table-item-pane> 592
<table-layout> 436
<text-cursor> 592
<text-editor> 592
<text-field> 593
<text-gadget> 595
<text-style> 162
<timer-event> 323
<tool-bar> 597
<top-level-sheet> 441
<transform> 57
<transform-error> 60
<transform-underspecified> 62
<tree-control> 598
<tree-node> 607
<tree-node-pane> 608
<undefined-text-style-mapping> 324
<value-gadget> 608
<value-range-gadget> 609
<viewport> 610
<>window-configuration-event> 324
<>window-event> 325

```

```

<window-repaint-event> 326
<wizard-frame> 751
<wizard-page> 754
clear-box generic function 207
clear-clipboard generic function 207
clear-progress-note generic
    function 630
client: init-keyword
    for <gadget> 485
clipboard-data-available? generic
    function 209
clipboard-owner generic function 210
clipboard-sheet generic function 209
clip-from-path generic function 356
close-clipboard function 210
close-path generic function 351, 356
<collection-gadget> class 478
<color> class 117
color: init-keyword
    for <color-not-found> 121
color? generic function 119
color-ihs generic function 120
color-luminosity generic function 121
<color-not-found> class 121
color-palette? generic function 122
color-rgb generic function 122
colors: init-keyword
    for <pattern> 152
<column-layout> class 401
columns: init-keyword
    for <button-box> 473
    for <table-layout> 436
    for <text-editor> 592
<combo-box> class 481
<command> class 630
command: init-keyword
    for <menu> 541
command? generic function 632
command-arguments generic function 632
command-enabled? generic function 632
command-enabled?-setter generic
    function 633
command-function generic function 634
command-queue: init-keyword
    for <simple-frame> 746
    for <simpleframe> 746
<command-table> class 634
command-table: init-keyword
    for <simple-frame> 746, 747
command-table? generic function 636
command-table-accelerators generic
    function 636
command-table-commands generic
    function 636
command-table-menu generic function 637
<command-table-menu-item> class 637
command-table-name generic function 639
command-undoable? generic function 640
complete-from-generator generic
    function 640
complete-from-sequence generic
    function 642
compose-rotation-with-transform
    generic function 19
compose-scaling-with-transform
    generic function 20
compose-space generic function 404
compose-transforms generic function 21
compose-transform-with-rotation
    generic function 22
compose-transform-with-scaling
    generic function 23
compose-transform-with-translation
    generic function 24
compose-translation-with-transform
    generic function 25
compute-next-page generic function 644
compute-previous-page generic
    function 645
condition: init-keyword
    for <port-terminated-event> 291
constants
    $altgr-key 186
    $alt-key 185
    $background 105
    $black 105
    $blue 106
    $boole-1 107
    $boole-2 107
    $boole-and 108
    $boole-andc1 110
    $boole-andc2 110
    $boole-c1 107
    $boole-c2 108
    $boole-clr 106
    $boole-eqv 109
    $boole-ior 108
    $boole-nand 109
    $boole-nor 109
    $boole-ord1 110
    $boole-orc2 111
    $boole-set 106
    $boole-xor 108
    $bricks-stipple 111

```

```

$capslock-key 189
$control-key 211
$cross-hatch 124
$cyan 125
$dash-dot-dot-pen 125
$dash-dot-pen 126
$dashed-pen 126
$diagonal-hatch-down 130
$diagonal-hatch-up 130
$dotted-pen 131
$everywhere 28
$fill 410
$foreground 134
$green 135
$hearts-stipple 135
$horizontal-hatch 135
$hyper-key 254
$identity-transform 28
$largest-coordinate 31
$left-button 257
$magenta 139
$meta-key 271
$middle-button 272
$modifier-keys 274
$nowhere 40
$numeric-argument-marker 738
$option-key 278
$parquet-stipple 152
$pointer-buttons 281
$red 159
$right-button 295
$shift-key 317
$smallest-coordinate 56
$solid-pen 160
$super-key 318
$tiles-stipple 170
$unsupplied-argument-marker 750
$vertical-hatch 170
$white 170
$xor-brush 171
$yellow 171
contain generic function 645
container: init-keyword
    for <top-level-sheet> 441
container-region: init-keyword
    for <top-level-sheet> 441
contents: init-keyword
    for <table-layout> 436
contract-node generic function 482
contrasting-colors-limit generic
    function 123
contrasting-dash-patterns-limit
    generic function 124
$ccontrol-key constant 211
coordinates: init-keyword
    for <polygon> 86
    for <polyline> 89
copy-area generic function 357
copy-from-pixmap generic function 358
copy-to-pixmap generic function 359
$cross-hatch constant 124
current-frame function 647
*current-frame* variable 647
current-page: init-keyword
    for <tab-control> 578
current-pane generic function 405
<cursor> type 211
cursor: init-keyword
    for <sheet> 300
cursor? generic function 212
cursor-active? generic function 212
cursor-active?-setter generic
    function 212
cursor-focus? generic function 212
cursor-focus?-setter generic
    function 212
curve-to generic function 352, 360
$cyan constant 125

```

## D

```

$dash-dot-dot-pen constant 125
$dash-dot-pen constant 126
$dashed-pen constant 126
dashes: init-keyword
    for <pen> 153
default?: init-keyword
    for <push-button> 553
    for <push-menu-button> 555
default-background generic function 127
default-background-setter generic
    function 127
default-foreground generic function 128
default-foreground-setter generic
    function 128
default-port function 212
default-port-setter function 213
default-text-style generic function 129
default-text-style-setter generic
    function 129
define command definition macro 647
define command-table definition
    macro 647
define frame definition macro 650
define pane definition macro 406

```

definition macros  
  define command 647  
  define command-table 647  
  define frame 650  
  define pane 406  
deiconify-frame generic function 655  
destroy-frame generic function 655  
destroy-frame?: init-keyword  
  for <frame-exit-event> 705  
destroy-pixmap generic function 361  
destroy-port generic function 213  
destroy-sheet generic function 214  
<device-event> class 214  
<device-font> class 129  
\$diagonal-hatch-down constant 130  
\$diagonal-hatch-up constant 130  
dialog-apply-button generic  
  function 656  
dialog-apply-button-setter generic  
  function 657  
dialog-apply-callback generic  
  function 657  
dialog-back-button generic function 658  
dialog-back-button-setter generic  
  function 659  
dialog-back-callback generic  
  function 659  
<dialog-cancel> class 660  
dialog-cancel-button generic  
  function 660  
dialog-cancel-button-setter generic  
  function 661  
dialog-cancel-callback generic  
  function 662  
dialog-cancel-callback-setter generic  
  function 663  
dialog-current-page generic  
  function 663  
dialog-current-page-setter generic  
  function 664  
dialog-exit-button generic function 664  
dialog-exit-button-setter generic  
  function 665  
dialog-exit-callback generic  
  function 666  
dialog-exit-callback-setter generic  
  function 667  
dialog-exit-enabled? generic  
  function 668  
dialog-exit-enabled?-setter generic  
  function 669  
<dialog-frame> class 670

dialog-help-button generic function 674  
dialog-help-button-setter generic  
  function 675  
dialog-help-callback generic  
  function 676  
dialog-next-button generic function 677  
dialog-next-button-setter generic  
  function 678  
dialog-next-callback generic  
  function 678  
dialog-next-enabled? generic  
  function 679  
dialog-next-enabled?-setter generic  
  function 680  
dialog-next-page generic function 681  
dialog-next-page-setter generic  
  function 682  
dialog-page-changed-callback generic  
  function 683  
dialog-page-changed-callback-setter generic function 684  
dialog-page-complete? generic  
  function 684  
dialog-page-complete?-setter generic  
  function 685  
dialog-pages generic function 686  
dialog-pages-setter generic  
  function 686  
dialog-previous-page generic  
  function 687  
dialog-previous-page-setter generic  
  function 688  
<display> class 215  
display generic function 216  
display: init-keyword  
  for <top-level-sheet> 441  
display? generic function 216  
display-depth generic function 217  
display-function: init-keyword  
  for <drawing-pane> 409  
  for <simple-pane> 426  
display-height generic function 217  
display-menu generic function 483  
display-mm-height generic function 218  
display-mm-width generic function 219  
display-orientation generic  
  function 219  
display-pixel-height generic  
  function 220  
display-pixels-per-point generic  
  function 220  
display-pixel-width generic

```

        function 221
display-progress-note generic
        function 689
display-units generic function 221
display-width generic function 222
do-allocate-space generic
        function 407
do-children-containing-position
        generic function 223
do-children-overlapping-region
        generic function 223
do-command-line-names generic
        function 689
do-command-table-accelerators
        generic function 689
do-command-table-commands generic
        function 689
do-command-table-menu-items
        generic function 689
do-compose-space generic function 408
do-coordinates function 25
document: init-keyword
        for <frame> 696
documentation: init-keyword
        for <gadget> 485
do-displays inline function 224
do-endpoint-coordinates function 26
do-frames generic function 224
do-noting-progress generic
        function 689
do-polygon-coordinates generic
        function 68
do-polygon-segments generic
        function 69
do-ports function 225
do-presentation-translators
        generic function 689
do-regions generic function 26
do-sheet-children generic
        function 225
do-sheet-tree generic function 226
$dotted-pen constant 131
<double-click-event> class 226
do-with-double-buffering generic
        function 362
do-with-drawing-options generic
        function 227
do-with-output-to-pixmap generic
        function 362
do-with-pointer-grabbed generic
        function 228
do-with-sheet-medium generic
        function 229
do-with-text-style generic function 229
do-with-transform generic function 230
draw-arrow generic function 363
draw-bezier-curve generic function 364
draw-circle generic function 366
draw-design generic function 70
draw-ellipse generic function 367
draw-image generic function 369
<drawing-pane> class 409
draw-line generic function 370
draw-lines generic function 371
draw-oval generic function 372
draw-pixmap generic function 373
draw-point generic function 374
draw-points generic function 375
draw-polygon generic function 376
draw-rectangle generic function 377
draw-rectangles generic function 379
draw-regular-polygon generic
        function 380
draw-text generic function 382
draw-triangle generic function 384
DUIM-DCs library 101
duim-dcs module 104
DUIM-Extended-Geometry library 67
duim-extended-geometry module 68
DUIM-Frames library 615
duim-frames module 621
DUIM-Gadgets library 445
duim-gadgets module 465
DUIM-Geometry library 7
duim-geometry module 9
DUIM-Graphics library 339
duim-graphics module 354
DUIM-Layouts library 395
duim-layouts module 399
DUIM-Sheets library 173
duim-sheets module 183

E
<ellipse> class 70
ellipse? generic function 72
ellipse-center-point generic
        function 72
ellipse-center-position generic
        function 73
ellipse-end-angle generic function 73
ellipse-radii generic function 74
ellipse-start-angle generic function 75
<elliptical-arc> class 75
elliptical-arc? generic function 76

```

**enabled?:** init-keyword  
 for <gadget> 486  
**end-angle:** init-keyword  
 for <ellipse> 70  
 for <elliptical-arc> 76  
**end-path** generic function 351, 386  
**end-x:** init-keyword  
 for <line> 77  
**end-y:** init-keyword  
 for <line> 77  
**equalize-heights?:** init-keyword  
 for <column-layout> 402  
 for <row-layout> 424  
**equalize-widths?:** init-keyword  
 for <column-layout> 402  
 for <row-layout> 424  
**even-scaling-transform?** generic  
 function 27  
**<event>** class 230  
**event?** generic function 231  
**event-button** generic function 231  
**event-character** generic function 232  
**event-destroy-frame?** generic  
 function 689  
**event-key-name** generic function 233  
**event-matches-gesture?** generic  
 function 233  
**event-modifier-state** generic  
 function 234  
**event-pointer** generic function 234  
**event-queue:** init-keyword  
 for <frame> 696  
**event-region** generic function 235  
**event-sheet** generic function 235  
**event-status-code** generic function 690  
**event-x** generic function 236  
**event-y** generic function 236  
**\$everywhere** constant 28  
**execute-command** generic function 690  
**exit-button:** init-keyword  
 for <dialog-frame> 670  
**exit-buttons-position:** init-keyword  
 for <dialog-frame> 671  
**exit-callback:** init-keyword  
 for <dialog-frame> 670  
**exit-dialog** generic function 691  
**exit-enabled?:** init-keyword  
 for <dialog-frame> 670  
**exit-frame** generic function 692  
**expand-node** generic function 483

**F**  
**family:** init-keyword  
 for <text-style> 162  
**\$fill** constant 410  
**fill-path** generic function 351, 387  
**fill-rule:** init-keyword  
 for <brush> 112, 141  
**fill-style:** init-keyword  
 for <brush> 112, 141  
**find-color** generic function 131  
**find-display** function 237  
**find-frame** function 693  
**find-frame-manager** function 237  
**find-item** generic function 484  
**find-node** generic function 485  
**find-port** function 238  
**fix-coordinate** function 28  
**fixed-height?:** init-keyword  
 for <frame> 696  
 for <layout> 414  
**<fixed-layout>** class 410  
**fixed-width?:** init-keyword  
 for <frame> 696  
 for <layout> 414  
**fixed-width-font?** generic function 239  
**<focus-gadget>** class 485  
**font-ascent** generic function 239  
**font-descent** generic function 240  
**font-height** generic function 240  
**font-metrics** generic function 241  
**font-name:** init-keyword  
 for <device-font> 130  
**font-width** generic function 242  
**force-display** generic function 242  
**\$foreground** constant 134  
**foreground:** init-keyword  
 for <brush> 111, 140  
 for <frame> 696  
 for <sheet> 300  
**<frame>** class 694  
**frame:** init-keyword  
 for <frame-event> 243  
 for <top-level-sheet> 441  
**frame?** generic function 699  
**frame-accelerators** generic function 699  
**frame-accelerators-setter** generic  
 function 700  
**frame-calling-frame** generic  
 function 700  
**frame-can-exit?** generic function 700  
**frame-command-queue** generic  
 function 701

```

frame-command-table generic
  function 701
frame-command-table-setter generic
  function 701
<frame-created-event> class 702
frame-default-button generic
  function 703
frame-default-button-setter
  generic function 703
<frame-destroyed-event> class 702
<frame-event> class 243
frame-event-queue generic
  function 704
<frame-exited-event> class 704
<frame-exit-event> class 705
frame-fixed-height? generic
  function 706
frame-fixed-width? generic
  function 707
<frame-focus-event> class 706
frame-icon generic function 707
frame-icon-setter generic
  function 708
frame-input-focus generic
  function 708
frame-input-focus-setter generic
  function 709
frame-layout generic function 709
frame-layout-setter generic
  function 710
<frame-manager> class 243
frame-manager generic function 244
frame-manager: init-keyword
  for <top-level-sheet> 441
frame-manager? generic function 245
frame-manager-frames generic
  function 245
frame-manager-palette generic
  function 246
frame-manager-palette-setter
  generic function 246
frame-mapped? generic function 710
frame-mapped?-setter generic
  function 712
<frame-mapped-event> class 711
frame-menu-bar generic function 713
frame-menu-bar-setter generic
  function 713
frame-mode generic function 714
frame-owner generic function 715
frame-palette generic function 715
frame-palette-setter generic
  function 715
frame-parent generic function 716
frame-position generic function 716
frame-resizable? generic function 717
frame-size generic function 717
frame-state generic function 718
frame-status-bar generic function 719
frame-status-bar-setter generic
  function 719
frame-status-message generic
  function 720
frame-status-message-setter generic
  function 720
frame-thread generic function 721
frame-title generic function 721
frame-title-setter generic function 722
frame-tool-bar generic function 722
frame-tool-bar-setter generic
  function 723
frame-top-level generic function 723
<frame-unmapped-event> class 724
fully-merged-text-style? generic
  function 134
function: init-keyword
  for <command> 631
  for <simple-command> 745
functions
  box-bottom 14
  box-edges 14
  box-height 15
  box-left 16
  box-position 16
  box-right 17
  box-size 18
  box-top 18
  box-width 19
  button-index-name 187
  close-clipboard 210
  current-frame 647
  default-port 212
  default-port-setter 213
  do-coordinates 25
  do-endpoint-coordinates 26
  do-ports 225
  find-display 237
  find-frame 693
  find-frame-manager 237
  find-port 238
  fix-coordinate 28
  gesture-spec-equal 249
  make-3-point-transform 31
  make-bounding-box 32

```

`make-command-table` 731  
`make-contrasting-colors` 142  
`make-contrasting-dash-patterns` 143  
`make-device-font` 143  
`make-ellipse` 80  
`make-elliptical-arc` 81  
`make-frame` 731  
`make-gray-color` 144  
`make-ihs-color` 145  
`make-line` 83  
`make-modifier-state` 258  
`make-pattern` 146  
`make-point` 33  
`make-polygon` 83  
`make-polyline` 84  
`make-rectangle` 85  
`make-reflection-transform` 34  
`make-rgb-color` 147  
`make-rotation-transform` 35  
`make-scaling-transform` 36  
`make-stencil` 148  
`make-text-style` 148  
`make-transform` 38  
`make-translation-transform` 39  
`modifier-key-index` 273  
`modifier-key-index-name` 273  
`remove-command-table` 743  
`unsupplied-argument?` 750

## G

`<gadget>` class 485  
`gadget?` generic function 488  
`gadget-accelerator` generic function 488  
`gadget-accelerator-setter` generic function 489  
`gadget-activate-callback` generic function 490  
`gadget-activate-callback-setter` generic function 490  
`gadget-armed-callback` generic function 491  
`gadget-armed-callback-setter` generic function 491  
`gadget-client` generic function 491  
`gadget-client-setter` generic function 492  
`gadget-command` generic function 492  
`gadget-command-setter` generic function 493  
`gadget-default?` generic function 493  
`gadget-default?-setter` generic function 494

`gadget-disarmed-callback` generic function 495  
`gadget-disarmed-callback-setter` generic function 495  
`gadget-documentation` generic function 495  
`gadget-documentation-setter` generic function 496  
`gadget-enabled?` generic function 496  
`gadget-enabled?-setter` generic function 497  
`gadget-focus-in-callback` generic function 498  
`gadget-focus-in-callback-setter` generic function 498  
`gadget-focus-out-callback` generic function 498  
`gadget-focus-out-callback-setter` generic function 498  
`gadget-id` generic function 498  
`gadget-id-setter` generic function 499  
`gadget-items` generic function 500  
`gadget-items-setter` generic function 501  
`gadget-key-press-callback` generic function 502  
`gadget-key-press-callback-setter` generic function 503  
`gadget-label` generic function 504  
`gadget-label-key` generic function 504  
`gadget-label-setter` generic function 505  
`gadget-mnemonic` generic function 506  
`gadget-mnemonic-setter` generic function 506  
`gadget-orientation` generic function 507  
`gadget-popup-menu-callback` generic function 508  
`gadget-popup-menu-callback-setter` generic function 508  
`gadget-read-only?` generic function 510  
`gadget-scrolling-horizontally?` generic function 510  
`gadget-scrolling-vertically?` generic function 510  
`gadget-selection` generic function 511  
`gadget-selection-mode` generic function 512  
`gadget-selection-setter` generic function 513  
`gadget-slug-size` generic function 514  
`gadget-slug-size-setter` generic

```

        function 515
gadget-test generic function 515
gadget-text generic function 516
gadget-text-setter generic
    function 517
gadget-value generic function 517
gadget-value-changed-callback
    generic function 519
gadget-value-changed-callback-
    setter generic function 519
gadget-value-changing-callback
    generic function 520
gadget-value-changing-callback-
    setter generic function 520
gadget-value-key generic function 521
gadget-value-range generic
    function 522
gadget-value-range-setter generic
    function 523
gadget-value-setter generic
    function 524
gadget-value-type generic
    function 525
gadget-x-alignment generic
    function 525
gadget-y-alignment generic
    function 526
generate-panes generic function 725
generation: init-keyword
    for <tree-node> 607
generator: init-keyword
    for <table-column> 585
generators: init-keyword
    for <table-control> 586
generic functions
    abort-path 351, 354
    activate-gadget 466
    add-child 184
    add-clipboard-data 184
    add-clipboard-data-as 185
    add-colors 105
    add-column 467
    add-command 621
    add-command-line-name 623
    add-command-table-menu-item 623
    add-item 467
    add-node 468
    add-presentation-translator 627
    allocate-space 399
    apply-in-frame 628
    arc-to 353, 354
    area? 11
    beep 186
    boundary-event-kind 186
    bounding-box 13
    bounding-box? 12
    brush? 112
    brush-background 113
    brush-fill-rule 113
    brush-fill-style 114
    brush-foreground 114
    brush-mode 114
    brush-stipple 115
    brush-stretch-mode 115
    brush-tile 116
    brush-ts-x 116
    brush-ts-y 117
    call-in-frame 628
    cancel-dialog 629
    caret-position 190
    caret-sheet 191
    caret-size 191
    caret-visible? 192
    caret-visible?-setter 192
    child-containing-position 193
    children-overlapping-region 193
    choose-color 194
    choose-directory 196
    choose-file 198
    choose-from-dialog 202
    choose-from-menu 204
    clear-box 207
    clear-clipboard 207
    clear-progress-note 630
    clipboard-data-available? 209
    clipboard-owner 210
    clipboard-sheet 209
    clip-from-path 356
    close-path 351, 356
    color? 119
    color-ihs 120
    color-luminosity 121
    color-palette? 122
    color-rgb 122
    command? 632
    command-arguments 632
    command-enabled? 632
    command-enabled?-setter 633
    command-function 634
    command-table? 636
    command-table-accelerators 636
    command-table-commands 636
    command-table-menu 637
    command-table-name 639

```

command-undoable? 640  
complete-from-generator 640  
complete-from-sequence 642  
compose-rotation-with-  
    transform 19  
compose-scaling-with-transform 20  
compose-space 404  
compose-transforms 21  
compose-transform-with-  
    rotation 22  
compose-transform-with-scaling 23  
compose-transform-with-  
    translation 24  
compose-translation-with-  
    transform 25  
compute-next-page 644  
compute-previous-page 645  
contain 645  
contract-node 482  
contrasting-colors-limit 123  
contrasting-dash-patterns-  
    limit 124  
copy-area 357  
copy-from-pixmap 358  
copy-to-pixmap 359  
current-pane 405  
cursor? 212  
cursor-active? 212  
cursor-active?-setter 212  
cursor-focus? 212  
cursor-focus?-setter 212  
curve-to 352, 360  
default-background 127  
default-background-setter 127  
default-foreground 128  
default-foreground-setter 128  
default-text-style 129  
default-text-style-setter 129  
deiconify-frame 655  
destroy-frame 655  
destroy-pixmap 361  
destroy-port 213  
destroy-sheet 214  
dialog-apply-button 656  
dialog-apply-button-setter 657  
dialog-apply-callback 657  
dialog-back-button 658  
dialog-back-button-setter 659  
dialog-back-callback 659  
dialog-cancel-button 660  
dialog-cancel-button-setter 661  
dialog-cancel-callback 662  
dialog-cancel-callback-setter 663  
dialog-current-page 663  
dialog-current-page-setter 664  
dialog-exit-button 664  
dialog-exit-button-setter 665  
dialog-exit-callback 666  
dialog-exit-callback-setter 667  
dialog-exit-enabled? 668  
dialog-exit-enabled?-setter 669  
dialog-help-button 674  
dialog-help-button-setter 675  
dialog-help-callback 676  
dialog-next-button 677  
dialog-next-button-setter 678  
dialog-next-callback 678  
dialog-next-enabled? 679  
dialog-next-enabled?-setter 680  
dialog-next-page 681  
dialog-next-page-setter 682  
dialog-page-changed-callback 683  
dialog-page-changed-callback-  
    setter 684  
dialog-page-complete? 684  
dialog-page-complete?-setter 685  
dialog-pages 686  
dialog-pages-setter 686  
dialog-previous-page 687  
dialog-previous-page-setter 688  
display 216  
display? 216  
display-depth 217  
display-height 217  
display-menu 483  
display-mm-height 218  
display-mm-width 219  
display-orientation 219  
display-pixel-height 220  
display-pixels-per-point 220  
display-pixel-width 221  
display-progress-note 689  
display-units 221  
display-width 222  
do-allocate-space 407  
do-children-containing-  
    position 223  
do-children-overlapping-region 223  
do-command-line-names 689  
do-command-table-accelerators 689  
do-command-table-commands 689  
do-command-table-menu-items 689  
do-compose-space 408  
do-frames 224

do-noting-progress 689  
do-polygon-coordinates 68  
do-polygon-segments 69  
do-presentation-translators 689  
do-regions 26  
do-sheet-children 225  
do-sheet-tree 226  
do-with-double-buffering 362  
do-with-drawing-options 227  
do-with-output-to-pixmap 362  
do-with-pointer-grabbed 228  
do-with-sheet-medium 229  
do-with-text-style 229  
do-with-transform 230  
draw-arrow 363  
draw-bezier-curve 364  
draw-circle 366  
draw-design 70  
draw-ellipse 367  
draw-image 369  
draw-line 370  
draw-lines 371  
draw-oval 372  
draw-pixmap 373  
draw-point 374  
draw-points 375  
draw-polygon 376  
draw-rectangle 377  
draw-rectangles 379  
draw-regular-polygon 380  
draw-text 382  
draw-triangle 384  
ellipse? 72  
ellipse-center-point 72  
ellipse-center-position 73  
ellipse-end-angle 73  
ellipse-radii 74  
ellipse-start-angle 75  
elliptical-arc? 76  
end-path 351, 386  
even-scaling-transform? 27  
event? 231  
event-button 231  
event-character 232  
event-destroy-frame? 689  
event-key-name 233  
event-matches-gesture? 233  
event-modifier-state 234  
event-pointer 234  
event-region 235  
event-sheet 235  
event-status-code 690  
event-x 236  
event-y 236  
execute-command 690  
exit-dialog 691  
exit-frame 692  
expand-node 483  
fill-path 351, 387  
find-color 131  
find-item 484  
find-node 485  
fixed-width-font? 239  
font-ascent 239  
font-descent 240  
font-height 240  
font-metrics 241  
font-width 242  
force-display 242  
frame? 699  
frame-accelerators 699  
frame-accelerators-setter 700  
frame-calling-frame 700  
frame-can-exit? 700  
frame-command-queue 701  
frame-command-table 701  
frame-command-table-setter 701  
frame-default-button 703  
frame-default-button-setter 703  
frame-event-queue 704  
frame-fixed-height? 706  
frame-fixed-width? 707  
frame-icon 707  
frame-icon-setter 708  
frame-input-focus 708  
frame-input-focus-setter 709  
frame-layout 709  
frame-layout-setter 710  
frame-manager 244  
frame-manager? 245  
frame-manager-frames 245  
frame-manager-palette 246  
frame-manager-palette-setter 246  
frame-mapped? 710  
frame-mapped?-setter 712  
frame-menu-bar 713  
frame-menu-bar-setter 713  
frame-mode 714  
frame-owner 715  
frame-palette 715  
frame-palette-setter 715  
frame-parent 716  
frame-position 716  
frame-resizable? 717

frame-size 717  
frame-state 718  
frame-status-bar 719  
frame-status-bar-setter 719  
frame-status-message 720  
frame-status-message-setter 720  
frame-thread 721  
frame-title 721  
frame-title-setter 722  
frame-tool-bar 722  
frame-tool-bar-setter 723  
frame-top-level 723  
fully-merged-text-style? 134  
gadget? 488  
gadget-accelerator 488  
gadget-accelerator-setter 489  
gadget-activate-callback 490  
gadget-activate-callback-  
    setter 490  
gadget-armed-callback 491  
gadget-armed-callback-setter 491  
gadget-client 491  
gadget-client-setter 492  
gadget-command 492  
gadget-command-setter 493  
gadget-default? 493  
gadget-default?-setter 494  
gadget-disarmed-callback 495  
gadget-disarmed-callback-  
    setter 495  
gadget-documentation 495  
gadget-documentation-setter 496  
gadget-enabled? 496  
gadget-enabled?-setter 497  
gadget-focus-in-callback 498  
gadget-focus-in-callback-  
    setter 498  
gadget-focus-out-callback 498  
gadget-focus-out-callback-  
    setter 498  
gadget-id 498  
gadget-id-setter 499  
gadget-items 500  
gadget-items-setter 501  
gadget-key-press-callback 502  
gadget-key-press-callback-  
    setter 503  
gadget-label 504  
gadget-label-key 504  
gadget-label-setter 505  
gadget-mnemonic 506  
gadget-mnemonic-setter 506  
gadget-orientation 507  
gadget-popup-menu-callback 508  
gadget-popup-menu-callback-  
    setter 508  
gadget-read-only? 510  
gadget-scrolling-horizontally? 510  
gadget-scrolling-vertically? 510  
gadget-selection 511  
gadget-selection-mode 512  
gadget-selection-setter 513  
gadget-slug-size 514  
gadget-slug-size-setter 515  
gadget-test 515  
gadget-text 516  
gadget-text-setter 517  
gadget-value 517  
gadget-value-changed-callback 519  
gadget-value-changed-callback-  
    setter 519  
gadget-value-changing-callback 520  
gadget-value-changing-callback-  
    setter 520  
gadget-value-key 521  
gadget-value-range 522  
gadget-value-range-setter 523  
gadget-value-setter 524  
gadget-value-type 525  
gadget-x-alignment 525  
gadget-y-alignment 526  
generate-panes 725  
gesture-button 247  
gesture-keysym 248  
gesture-modifier-state 248  
get-default-background 250  
get-default-foreground 250  
get-default-text-style 251  
handle-event 252  
handle-repaint 253  
iconify-frame 725  
identity-transform? 29  
image? 136  
image-depth 137  
image-height 137  
image-width 138  
ink? 139  
invertible-transform? 30  
invert-transform 29  
item-object 528  
layout-border 415  
layout-border-setter 415  
layout-equalize-heights? 416, 417  
layout-frame 726

line? 78  
line-end-point 78  
line-end-position 78  
line-start-point 79  
line-start-position 79  
line-to 352, 387  
list-control-icon-function 533  
list-control-icon-function-setter 534  
list-control-view 535  
list-control-view-setter 536  
lower-frame 727  
lower-progress-note 728  
lower-sheet 257  
make-color-for-contrasting-color 141  
make-container 731  
make-frame-manager 258  
make-item 538  
make-menu-from-command-table-menu 731  
make-menu-from-items 539  
make-menus-from-command-table 732  
make-node 540  
make-palette 145  
make-pane 259  
make-pixmap 388  
medium? 261  
medium-background 261  
medium-background-setter 262  
medium-brush 262  
medium-brush-setter 263  
medium-clipping-region 263  
medium-clipping-region-setter 264  
medium-default-text-style 264  
medium-default-text-style-setter 265  
medium-drawable 265  
medium-drawable-setter 266  
medium-foreground 266  
medium-foreground-setter 266  
medium-merged-text-style 267  
medium-pen 267  
medium-pen-setter 268  
medium-pixmap 268  
medium-pixmap-setter 269  
medium-sheet 269  
medium-text-style 270  
medium-text-style-setter 270  
medium-transform 271  
medium-transform-setter 271  
menu-item-accelerator 733  
menu-item-mnemonic 733  
menu-item-name 734  
menu-item-options 734  
menu-item-type 735  
menu-item-value 735  
menu-owner 545  
merge-text-styles 150  
move-to 352, 389  
move-to-next-page 736  
move-to-previous-page 736  
node-children 546  
node-children-setter 546  
node-expanded? 547  
node-object 547  
node-parents 548  
node-state 548  
note-command-table-changed 737  
note-frame-terminated 737  
note-port-terminated 275  
note-progress 737  
note-progress-in-phases 738  
note-viewport-position-changed 549  
note-viewport-region-changed 549  
notify-user 275  
palette? 151  
pane-display-function 420  
pane-layout 421  
partial-command? 738  
path? 41  
pattern? 153  
pen? 154  
pen-cap-shape 155  
pen-dashes 155  
pen-joint-shape 156  
pen-units 157  
pen-width 158  
pixmap? 391  
point? 42  
pointer? 279  
pointer-button-state 282  
pointer-cursor 282  
pointer-cursor-setter 282  
pointer-position 286  
pointer-sheet 287  
point-position 42  
point-x 43  
point-y 43  
polygon? 88  
polygon-coordinates 88  
polygon-points 89

polyline? 91  
polyline-closed? 91  
port 288  
port? 289  
port-modifier-state 289  
port-name 290  
port-pointer 290  
port-server-path 291  
port-type 292  
progress-note-label 739  
progress-note-label-setter 739  
queue-event 292  
queue-repaint 293  
raise-frame 741  
raise-progress-note 742  
raise-sheet 293  
read-image 158  
read-image-as 159  
rectangle? 93  
rectangle-edges 94  
rectangle-height 95  
rectangle-max-point 96  
rectangle-max-position 96  
rectangle-min-point 97  
rectangle-min-position 98  
rectangle-size 99  
rectangle-width 99  
rectilinear-transform? 44  
redo-command 742  
reflection-transform? 44  
region? 46  
region-contains-position? 46  
region-contains-region? 47  
region-difference 48  
region-empty? 48  
region-equal 49  
region-intersection 49  
region-intersects-region? 50  
region-set? 51  
region-set-function 51  
region-set-regions 52  
region-union 53  
relayout-children 422  
relayout-parent 423  
remove-child 294  
remove-colors 160  
remove-column 559  
remove-command 743  
remove-command-line-name 743  
remove-command-table-menu-item 744  
remove-item 560  
remove-node 560  
remove-presentation-translator 744  
repaint-sheet 294  
replace-child 295  
restart-port 295  
restore-clipping-region 392  
rigid-transform? 53  
run-frame-top-level 744  
save-clipping-region 392  
scaling-transform? 54  
scroll-position 564  
set-box-edges 54  
set-box-position 55  
set-box-size 55  
set-caret-position 296  
set-frame-position 744  
set-frame-size 745  
set-pointer-position 296  
set-scroll-position 565  
set-sheet-edges 297  
set-sheet-position 297  
set-sheet-size 298  
sheet? 303  
sheet-ancestor? 304  
sheet-child 304  
sheet-children 305  
sheet-children-setter 305  
sheet-child-setter 306  
sheet-edges 306  
sheet-event-mask 308  
sheet-event-mask-setter 308  
sheet-event-queue 309  
sheet-frame 309  
sheet-mapped? 309  
sheet-mapped?-setter 310  
sheet-medium 310  
sheet-parent 311  
sheet-parent-setter 311  
sheet-pointer-cursor 312  
sheet-pointer-cursor-setter 312  
sheet-position 313  
sheet-region 313  
sheet-region-setter 314  
sheet-size 314  
sheet-state 315  
sheet-text-cursor 315  
sheet-transform 316  
sheet-transform-setter 317  
sheet-viewport 566  
sheet-viewport-region 567  
sheet-withdrawn? 317  
space-requirement? 429  
space-requirement-height 430

space-requirement-max-height 430  
 space-requirement-max-width 431  
 space-requirement-min-height 431  
 space-requirement-min-width 432  
 space-requirement-width 432  
 splitter-split-bar-moved-  
     callback 572, 573  
 splitter-split-bar-moved-call-  
     back-setter 572, 573  
 splitter-split-box-callback 572,  
     573  
 splitter-split-box-callback-  
     setter 573  
 stack-layout-mapped-page 434  
 stack-layout-mapped-page-  
     getter 435  
 start-dialog 748  
 start-frame 748  
 start-path 392  
 status-bar-label-pane 576  
 status-bar-label-pane-setter 577  
 status-bar-progress-bar 577  
 status-bar-progress-bar-  
     setter 577  
 stencil? 161  
 stroke-path 352, 393  
 substitute-numeric-argument-  
     marker 750  
 substitute-unsupplied-argument-  
     marker 750  
 synchronize-display 319  
 tab-control-current-page 579  
 tab-control-current-page-  
     setter 580  
 tab-control-labels 581  
 tab-control-pages 583  
 tab-control-pages-setter 584  
 table-contents 435  
 table-contents-setter 435  
 table-control-view 591  
 table-control-view-setter 591  
 text-size 319  
 text-style? 164  
 text-style-components 164  
 text-style-family 165  
 text-style-mapping 321  
 text-style-mapping-exists? 321  
 text-style-mapping-setter 322  
 text-style-size 166  
 text-style-slant 167  
 text-style-strikeout? 168  
 text-style-underline? 168  
 text-style-weight 169  
 top-level-sheet 323  
 transform? 58  
 transform-angles 58  
 transform-box 59  
 transform-distance 60  
 transform-position 61  
 transform-region 61  
 translation-transform? 62  
 tree-control-children-  
     generator 602  
 tree-control-children-generator-  
     setter 603  
 tree-control-children-  
     predicate 601  
 tree-control-children-predicate-  
     setter 602  
 tree-control-icon-function 603  
 tree-control-initial-depth 604  
 tree-control-initial-depth-  
     setter 605  
 tree-control-roots 605  
 tree-control-roots-setter 606  
 undo-command 750  
 untransform-angles 63  
 untransform-box 63  
 untransform-distance 64  
 untransform-position 65  
 untransform-region 65  
 update-gadget 608  
 viewport? 612  
 viewport-region 612  
 write-image 171  
 <gesture> class 247  
 gesture-button generic function 247  
 gesture-keysym generic function 248  
 gesture-modifier-state generic  
     function 248  
 gesture-spec-equal function 249  
 get-default-background generic  
     function 250  
 get-default-foreground generic  
     function 250  
 get-default-text-style generic  
     function 251  
 \*global-command-table\* variable 725  
 \$green constant 135  
 green: init-keyword  
     for <color> 117  
 <grid-layout> class 411  
 <group-box> class 526

## H

handle-event generic function 252  
handle-repaint generic function 253  
heading: init-keyword  
    for <table-column> 585  
headings: init-keyword  
    for <table-control> 586  
\$hearts-stipple constant 135  
height: init-keyword  
    for <caret> 189  
    for <layout> 413  
    for <space-requirement> 427  
help-button: init-keyword  
    for <dialog-frame> 671  
help-callback: init-keyword  
    for <dialog-frame> 670  
help-context: init-keyword  
    for <basic-user-pane> 400  
    for <sheet> 299  
help-source: init-keyword  
    for <basic-user-pane> 400  
    for <sheet> 299  
\$horizontal-hatch constant 135  
horizontally statement macro 412  
horizontal-scroll-bar: init-keyword  
    for <viewport> 610  
horizontal-split-box?: init-keyword  
    for <splitter> 572  
hue: init-keyword  
    for <color> 117  
\$hyper-key constant 254

## I

icon: init-keyword  
    for <frame> 695  
icon-function: init-keyword  
    for <list-control> 531  
    for <tree-control> 598  
iconify-frame generic function 725  
id: init-keyword  
    for <gadget> 485  
\$identity-transform constant 28  
identity-transform? generic function 29  
<image> class 136  
image: init-keyword  
    for <command-table-menu-item> 638  
image? generic function 136  
image-depth generic function 137  
image-height generic function 137  
image-width generic function 138  
inherit-from: init-keyword

    for <command-table> 634  
inhibit-updating-scroll-bars state-  
    ment macro 528  
initial-depth: init-keyword  
    for <tree-control> 598  
init-keywords  
    402  
accelerator:  
    for <button> 470  
    for <command-table-menu-item> 638  
accepts-focus?:  
    for <sheet> 300  
activate-callback:  
    for <action-gadget> 466  
alignment:  
    for <table-column> 585  
apply-button:  
    for <property-frame> 739  
    for <wizard-frame> 751  
apply-callback:  
    for <property-frame> 739  
    for <wizard-frame> 751  
arguments:  
    for <command> 631  
    for <simple-command> 745  
array:  
    for <stencil> 161  
auto-scroll?:  
    for <text-field> 594  
background:  
    for <brush> 111, 140  
    for <frame> 696  
background:  
    for <sheet> 300  
blue:  
    for <color> 117  
border:  
    for <column-layout> 401  
    for <row-layout> 423  
    for <stack-layout> 433  
    for <table-layout> 436  
borders:  
    for <combo-box> 481  
    for <list-control> 531  
    for <option-box> 549  
    for <slider> 567  
    for <spin-box> 570  
    for <table-control> 586  
bottom:  
    for <bounding-box> 12  
button:  
    for <gesture> 247

```

for <pointer-button-event> 280
for <pointer-drag-event> 283
for <pointer-gesture> 285
callback:
for <table-column> 585
calling-frame:
for <frame> 695
cancel-button:
for <dialog-frame> 670
cancel-callback:
for <dialog-frame> 670
cap-shape:
for <pen> 153
caret:
for <sheet> 300
case:
for <text-field> 594
cell-space-requirement:
for <grid-layout> 411
center-point:
for <ellipse> 70
for <elliptical-arc> 75
center-x:
for <ellipse> 70
for <elliptical-arc> 75
center-y:
for <ellipse> 70
for <elliptical-arc> 75
character:
for <keyboard-event> 255
child:
for <button-box> 474
for <sheet> 299
for <spacing> 569
child-nodes:
for <tree-node> 607
children:
for <sheet> 299
children-generator:
for <tree-control> 598
children-predicate:
for <tree-control> 598
client:
for <gadget> 485
color:
for <color-not-found> 121
colors:
for <pattern> 152
columns:
for <button-box> 473
for <table-layout> 436
for <text-editor> 592
command:
for <menu> 541
command-queue:
for <simple-frame> 746
command-table:
for <simple-frame> 746, 747
condition:
for <port-terminated-event> 291
container:
for <top-level-sheet> 441
container-region:
for <top-level-sheet> 441
contents:
for <table-layout> 436
coordinates:
for <polygon> 86
for <polyline> 89
cursor:
for <sheet> 300
dashes:
for <pen> 153
default?:
for <push-button> 553
for <push-menu-button> 555
destroy-frame?:
for <frame-exit-event> 705
display:
for <top-level-sheet> 441
display-function:
for <drawing-pane> 409
for <simple-pane> 426
document:
for <frame> 696
documentation:
for <gadget> 485
enabled?:
for <gadget> 486
end-angle:
for <ellipse> 70
for <elliptical-arc> 76
end-x:
for <line> 77
end-y:
for <line> 77
equalize-heights?:
for <column-layout> 402
for <row-layout> 424
equalize-widths?:
for <column-layout> 402
for <row-layout> 424
event-queue:
for <frame> 696

```

```

exit-button:
  for <dialog-frame> 670
exit-buttons-position:
  for <dialog-frame> 671
exit-callback:
  for <dialog-frame> 670
exit-enabled?:
  for <dialog-frame> 670
family:
  for <text-style> 162
fill-rule:
  for <brush> 112, 141
fill-style:
  for <brush> 112, 141
fixed-height?:
  for <frame> 696
  for <layout> 414
fixed-width?:
  for <frame> 696
  for <layout> 414
font-name:
  for <device-font> 130
foreground:
  for <brush> 111, 140
  for <frame> 696
  for <sheet> 300
frame:
  for <frame-event> 243
  for <top-level-sheet> 441
frame-manager:
  for <top-level-sheet> 441
function:
  for <command> 631
  for <simple-command> 745
generation:
  for <tree-node> 607
generator:
  for <table-column> 585
generators:
  for <table-control> 586
green:
  for <color> 117
heading:
  for <table-column> 585
headings:
  for <table-control> 586
height:
  for <caret> 189
  for <layout> 413
  for <space-requirement> 427
help-button:
  for <dialog-frame> 671
help-callback:
  for <dialog-frame> 670
help-context:
  for <basic-user-pane> 400
  for <sheet> 299
help-source:
  for <basic-user-pane> 400
  for <sheet> 299
horizontal-scroll-bar:
  for <viewport> 610
horizontal-split-box?:
  for <splitter> 572
hue:
  for <color> 117
icon:
  for <frame> 695
icon-function:
  for <list-control> 531
  for <tree-control> 598
id:
  for <gadget> 485
image:
  for <command-table-menu-item> 638
inherit-from:
  for <command-table> 634
initial-depth:
  for <tree-control> 598
input-focus:
  for <frame> 696
intensity:
  for <color> 117
joint-shape:
  for <pen> 153
key-name:
  for <keyboard-event> 255
key-press-callback:
  for <collection-gadget> 479
  for <list-control> 531
  for <tab-control> 578
  for <table-control> 587
  for <tree-control> 598
keysym:
  for <gesture> 247
  for <keyboard-gesture> 255
kind:
  for <pointer-boundary-event> 280
label:
  for <gadget> 485
  for <group-box> 526
  for <label> 528
  for <page> 550
  for <space-requirement> 428

```

```

        for <status-bar> 573
label-key:
    for <collection-gadget> 478
label-pane:
    for <status-bar> 573
label-position:
    for <group-box> 527
layout:
    for <frame> 695
    for <simple-frame> 746, 747
layout-class:
    for <button-box> 473
left:
    for <bounding-box> 11
lines:
    for <text-editor> 592
max-height:
    for <layout> 413
    for <space-requirement> 428
max-label:
    for <slider> 567
max-width:
    for <layout> 413
    for <space-requirement> 427
max-x:
    for <rectangle> 92
max-y:
    for <rectangle> 92
menu-bar:
    for <simple-frame> 746, 747
min-height:
    for <layout> 413
    for <space-requirement> 427
min-label:
    for <slider> 567
min-width:
    for <layout> 413
    for <space-requirement> 427
min-x:
    for <rectangle> 92
min-y:
    for <rectangle> 92
mnemonic:
    for <button> 470
    for <command-table-menu-item> 638
    for <menu> 541
mode:
    for <brush> 112, 140
    for <dialog-frame> 670
modifiers:
    for <gesture> 247
modifier-state:
    for <device-event> 214
    for <gesture> 247
    for <keyboard-gesture> 255
    for <pointer-gesture> 285
mxx:
    for <transform> 57
mxy:
    for <transform> 57
myx:
    for <transform> 57
myy:
    for <transform> 57
name:
    for <command-table> 634
    for <command-table-menu-item> 637
object:
    for <list-item> 537
    for <table-item> 592
    for <tree-node> 607
opacity:
    for <color> 117
options:
    for <command-table-menu-item> 638
orientation:
    for <button-box> 473
    for <display> 215
    for <progress-bar> 551
    for <scroll-bar> 561
    for <separator> 565
    for <slider> 568
owner:
    for <menu> 541
page-changed-callback:
    for <dialog-frame> 671
    for <property-frame> 739
    for <wizard-frame> 751
pages:
    for <dialog-frame> 671
    for <property-frame> 739
    for <tab-control> 578
    for <wizard-frame> 751
palette:
    for <frame> 696
    for <palette-full> 151
parent:
    for <sheet> 299
parent-nodes:
    for <tree-node> 607
pointer:
    for <pointer-event> 284
points:

```

```

for <line> 77
for <polygon> 87
for <polyline> 90
for <rectangle> 92
for <reflection-
    underspecified> 45
for <transform-underspecified> 62
popup-menu-callback:
    for <list-control> 531
    for <table-control> 587
    for <tree-control> 598
port:
    for <basic-user-pane> 400
    for <device-font> 130
    for <pointer> 279
    for <sheet> 299
    for <undefined-text-style-
        mapping> 324
progress-bar:
    for <status-bar> 573
progress-bar?:
    for <status-bar> 573
radius-1-dx:
    for <ellipse> 70
    for <elliptical-arc> 75
radius-1-dy:
    for <ellipse> 70
    for <elliptical-arc> 75
radius-2-dx:
    for <ellipse> 70
    for <elliptical-arc> 75
radius-2-dy:
    for <ellipse> 70
    for <elliptical-arc> 75
ratios:
    for <column-layout> 402
    for <row-layout> 424
read-only?:
    for <gadget> 486
red:
    for <color> 117
region:
    for <basic-user-pane> 400
    for <sheet> 299
    for <window-event> 325
resizable?:
    for <frame> 696
resizeable?:
    for <layout> 414
resource-id:
    for <frame> 696
right:
    for <bounding-box> 11
roots:
    for <tree-control> 599
rows:
    for <button-box> 473
    for <table-layout> 436
saturation:
    for <color> 117
scroll-bars:
    for <combo-box> 481
    for <list-box> 530
    for <list-control> 531
    for <option-box> 549
    for <table-control> 587
    for <text-editor> 592
    for <tree-control> 598
selection:
    for <collection-gadget> 479
selection-mode:
    for <collection-gadget> 479
sheet:
    for <caret> 189
    for <device-event> 214
    for <sheet-event> 307
show-buttons?:
    for <tree-control> 598
show-edges?:
    for <tree-control> 598
show-root-edges?:
    for <tree-control> 598
size:
    for <text-style> 162
slant:
    for <text-style> 162
slug-size:
    for <scroll-bar> 561
space-requirement:
    for <layout> 413
spacing:
    for <column-layout> 402
split-bar-moved-callback:
    for <splitter> 571
split-box-callback:
    for <splitter> 571
start-angle:
    for <ellipse> 70
    for <elliptical-arc> 75
start-x:
    for <line> 77
start-y:
    for <line> 77
state:

```

```

for <frame> 695
status-bar:
    for <simple-frame> 746, 747
status-code:
    for <frame-exited-event> 704
stipple:
    for <brush> 112, 141
stretchable?:
    for <pinboard-layout> 421
strikeout?:
    for <text-style> 162
style-descriptor:
    for <basic-user-pane> 400
    for <sheet> 299
test:
    for <collection-gadget> 479
text:
    for <text-gadget> 595
text-style:
    for <frame> 696
    for <sheet> 300
    for <undefined-text-style-
        mapping> 324
thickness:
    for <border> 469
    for <spacing> 569
thread:
    for <frame> 695
tick-marks:
    for <slider> 567
tile:
    for <brush> 112, 141
timestamp:
    for <event> 230
title:
    for <frame> 695
tool-bar:
    for <simple-frame> 746, 747
top:
    for <bounding-box> 11
top-level-sheet:
    for <frame> 695
transform:
    for <basic-user-pane> 400
    for <sheet> 299
    for <singular-transform> 56
    for <stencil> 161
ts-x:
    for <brush> 112, 141
ts-y:
    for <brush> 112, 141
tx:
    for <transform> 57
ty:
    for <transform> 57
type:
    for <border> 469
    for <command-table-menu-item> 638
underline?:
    for <text-style> 162
undo-command:
    for <simple-undoable-command> 747
units:
    for <display> 215
    for <pen> 153
update-callback:
    for <menu> 541
    for <menu-bar> 542
    for <menu-box> 543
    for <menu-button> 544
    for <tool-bar> 597
value:
    for <command-table-menu-item> 638
    for <status-bar> 573
    for <value-gadget> 609
value-changed-callback:
    for <scroll-bar> 561
    for <value-gadget> 609
value-changing-callback:
    for <scroll-bar> 561
    for <slider> 568
    for <text-gadget> 595
value-key:
    for <collection-gadget> 479
value-range:
    for <status-bar> 573
    for <value-range-gadget> 610
value-type:
    for <text-gadget> 595
vertical-scroll-bar:
    for <viewport> 611
vertical-split-box?:
    for <splitter> 572
view:
    for <list-control> 531
    for <table-control> 586
visible-child:
    for <tab-control> 578
weight:
    for <text-style> 162
width:
    for <caret> 189
    for <layout> 413
    for <pen> 153

```

```

for <space-requirement> 427
for <table-column> 585
widths:
  for <table-control> 587
withdrawn:
  for <sheet> 299
x:
  for <caret> 189
  for <point> 41
  for <pointer-event> 284
  for <sheet> 299
x-alignment:
  for <column-layout> 402
  for <table-layout> 437
  for <text-field> 594
x-ratios: 424
  for <table-layout> 436
x-spacing:
  for <row-layout> 424
  for <table-layout> 436
y:
  for <caret> 189
  for <point> 41
  for <pointer-event> 284
  for <sheet> 299
y-alignment:
  for <row-layout> 424
  for <table-layout> 437
y-r 402
y-ratios:
  for <table-layout> 436
y-spacing:
  for <table-layout> 436
zzz 424
<ink> class 138
ink? generic function 139
inline functions
  button-index 187
  do-displays 224
input-focus: init-keyword
  for <frame> 696
intensity: init-keyword
  for <color> 117
invertible-transform? generic
  function 30
invert-transform generic function 29
item-object generic function 528

```

## J

joint-shape: init-keyword  
 for <pen> 153

**K**

<keyboard-event> class 254  
<keyboard-gesture> class 255  
key-name: init-keyword  
 for <keyboard-event> 255  
key-press-callback: init-keyword  
 for <collection-gadget> 479  
 for <list-control> 531  
 for <tab-control> 578  
 for <table-control> 587  
 for <tree-control> 598  
<key-press-event> class 256  
<key-release-event> class 256  
keysym: init-keyword  
 for <gesture> 247  
 for <keyboard-gesture> 255  
kind: init-keyword  
 for <pointer-boundary-event> 280

## L

<label> class 528  
label: init-keyword  
 for <gadget> 485  
 for <group-box> 526  
 for <label> 528  
 for <page> 550  
 for <space-requirement> 428  
 for <status-bar> 573  
label-key: init-keyword  
 for <collection-gadget> 478  
labelling statement macro 529  
label-pane: init-keyword  
 for <status-bar> 573  
label-position: init-keyword  
 for <group-box> 527  
\$largest-coordinate constant 31  
<layout> class 413  
layout: init-keyword  
 for <frame> 695  
 for <simple-frame> 746, 747  
layout-border generic function 415  
layout-border-setter generic
 function 415  
layout-class: init-keyword  
 for <button-box> 473  
layout-equalize-heights? generic
 function 416, 417  
layout-frame generic function 726  
<leaf-pane> class 417  
left: init-keyword  
 for <bounding-box> 11

**\$left-button** constant 257  
**libraries**  
 DUIM-DCs 101  
 DUIM-Extended-Geometry 67  
 DUIM-Frames 615  
 DUIM-Gadgets 445  
 DUIM-Geometry 7  
 DUIM-graphics 339  
 DUIM-layouts 395  
 DUIM-Sheets 173  
**<line>** class 77  
**line?** generic function 78  
**line-end-point** generic function 78  
**line-end-position** generic function 78  
**lines:** init-keyword  
 for **<text-editor>** 592  
**line-start-point** generic function 79  
**line-start-position** generic  
 function 79  
**line-to** generic function 352, 387  
**<list-box>** class 529  
**<list-control>** class 531  
**list-control-icon-function** generic  
 function 533  
**list-control-icon-function-setter**  
 generic function 534  
**list-control-view** generic  
 function 535  
**<list-control-view>** type 534  
**list-control-view-setter** generic  
 function 536  
**<list-item>** class 537  
**<list-item-pane>** class 537  
**lower-frame** generic function 727  
**lower-progress-note** generic  
 function 728  
**lower-sheet** generic function 257

**M**

**\$magenta** constant 139  
**make** method 140, 418, 728  
**make-3-point-transform** function 31  
**make-bounding-box** function 32  
**make-color-for-contrasting-color**  
 generic function 141  
**make-command-table** function 731  
**make-container** generic function 731  
**make-contrasting-colors**  
 function 142  
**make-contrasting-dash-patterns**  
 function 143  
**make-device-font** function 143

**make-ellipse** function 80  
**make-elliptical-arc** function 81  
**make-frame** function 731  
**make-frame-manager** generic function 258  
**make-gray-color** function 144  
**make-ihs-color** function 145  
**make-item** generic function 538  
**make-line** function 83  
**make-menu-from-command-table-menu**  
 generic function 731  
**make-menu-from-items** generic  
 function 539  
**make-menus-from-command-table** generic  
 function 732  
**make-modifier-state** function 258  
**make-node** generic function 540  
**make-palette** generic function 145  
**make-pane** generic function 259  
**make-pattern** function 146  
**make-pixmap** generic function 388  
**make-point** function 33  
**make-polygon** function 83  
**make-polyline** function 84  
**make-rectangle** function 85  
**make-reflection-transform** function 34  
**make-rgb-color** function 147  
**make-rotation-transform** function 35  
**make-scaling-transform** function 36  
**make-stencil** function 148  
**make-text-style** function 148  
**make-transform** function 38  
**make-translation-transform**  
 function 39  
**max-height:** init-keyword  
 for **<layout>** 413  
 for **<space-requirement>** 428  
**max-label:** init-keyword  
 for **<slider>** 567  
**max-width:** init-keyword  
 for **<layout>** 413  
 for **<space-requirement>** 427  
**max-x:** init-keyword  
 for **<rectangle>** 92  
**max-y:** init-keyword  
 for **<rectangle>** 92  
**<medium>** class 259  
**medium?** generic function 261  
**medium-background** generic function 261  
**medium-background-setter** generic  
 function 262  
**medium-brush** generic function 262  
**medium-brush-setter** generic

```

        function 263
medium-clipping-region generic
        function 263
medium-clipping-region-setter generic
        function 264
medium-default-text-style generic
        function 264
medium-default-text-style-setter
        generic function 265
medium-drawable generic function 265
medium-drawable-setter generic
        function 266
medium-foreground generic function 266
medium-foreground-setter generic
        function 266
medium-merged-text-style generic
        function 267
medium-pen generic function 267
medium-pen-setter generic function 268
medium-pixmap generic function 268
medium-pixmap-setter generic
        function 269
medium-sheet generic function 269
medium-text-style generic function 270
medium-text-style-setter generic
        function 270
medium-transform generic function 271
medium-transform-setter generic
        function 271
<menu> class 541
<menu-bar> class 542
menu-bar: init-keyword
    for <simple-frame> 746, 747
<menu-box> class 543
<menu-button> class 544
menu-item-accelerator generic
        function 733
menu-item-mnemonic generic function 733
menu-item-name generic function 734
menu-item-options generic function 734
menu-item-type generic function 735
menu-item-value generic function 735
menu-owner generic function 545
merge-text-styles generic function 150
$meta-key constant 271
methods
    = 10, 104, 183, 621
    make 140, 418, 728
$middle-button constant 272
min-height: init-keyword
    for <layout> 413
    for <space-requirement> 427
min-label: init-keyword
    for <slider> 567
min-width: init-keyword
    for <layout> 413
    for <space-requirement> 427
min-x: init-keyword
    for <rectangle> 92
min-y: init-keyword
    for <rectangle> 92
mnemonic: init-keyword
    for <button> 470
    for <command-table-menu-item> 638
    for <menu> 541
mode: init-keyword
    for <brush> 112, 140
    for <dialog-frame> 670
modifier-key-index function 273
modifier-key-index-name function 273
$modifier-keys constant 274
modifiers: init-keyword
    for <gesture> 247
modifier-state: init-keyword
    for <device-event> 214
    for <gesture> 247
    for <keyboard-gesture> 255
    for <pointer-gesture> 285
modules
    duim-dcs 104
    duim-extended-geometry 68
    duim-frames 621
    duim-gadgets 465
    duim-geometry 9
    duim-graphics 354
    duim-layouts 399
    duim-silica 183
move-to generic function 352, 389
move-to-next-page generic function 736
move-to-previous-page generic
        function 736
<multiple-child-composite-pane>
    class 419
mxz: init-keyword
    for <transform> 57
myx: init-keyword
    for <transform> 57
myx: init-keyword
    for <transform> 57
myy: init-keyword
    for <transform> 57

```

## N

**name**: init-keyword

```

for <command-table> 634
  for <command-table-menu-item> 637
node-children generic function 546
node-children-setter generic
  function 546
node-expanded? generic function 547
node-object generic function 547
node-parents generic function 548
node-state generic function 548
note-command-table-changed generic
  function 737
note-frame-terminated generic
  function 737
note-port-terminated generic
  function 275
note-progress generic function 737
note-progress-in-phases generic
  function 738
note-viewport-position-changed
  generic function 549
note-viewport-region-changed
  generic function 549
notify-user generic function 275
noting-progress statement macro 738
$nowhere constant 40
<null-pane> class 420
$numeric-argument-marker
  constant 738
numtick-marks: init-keyword
  for <slider> 567

```

## O

```

object: init-keyword
  for <list-item> 537
  for <table-item> 592
  for <tree-node> 607
OLE objects 441
opacity: init-keyword
  for <color> 117
<option-box> class 549
$option-key constant 278
options: init-keyword
  for <command-table-menu-item> 638
orientation: init-keyword
  for <button-box> 473
  for <display> 215
  for <progress-bar> 551
  for <scroll-bar> 561
  for <separator> 565
  for <slider> 568
owner: init-keyword
  for <menu> 541

```

## P

```

<page> class 550
page-changed-callback: init-keyword
  for <dialog-frame> 671
  for <property-frame> 739
  for <wizard-frame> 751
pages: init-keyword
  for <dialog-frame> 671
  for <property-frame> 739
  for <tab-control> 578
  for <wizard-frame> 751
<palette> class 150
palette: init-keyword
  for <frame> 696
  for <palette-full> 151
palette? generic function 151
<palette-full> class 151
pane-display-function generic
  function 420
pane-layout generic function 421
parent: init-keyword
  for <sheet> 299
parent-nodes: init-keyword
  for <tree-node> 607
$parquet-stipple constant 152
partial-command? generic function 738
<password-field> class 550
<path> class 40
path? generic function 41
<pattern> class 152
pattern? generic function 153
<pen> class 153
pen? generic function 154
pen-cap-shape generic function 155
pen-dashes generic function 155
pen-joint-shape generic function 156
pen-units generic function 157
pen-width generic function 158
<pinboard-layout> class 421
<pixmap> class 390
pixmap? generic function 391
<pixmap-medium> class 391
<point> class 41
point? generic function 42
<pointer> class 278
pointer: init-keyword
  for <pointer-event> 284
pointer? generic function 279
<pointer-boundary-event> class 279
<pointer-button-event> class 280
$pointer-buttons constant 281
pointer-button-state generic

```

```

        function 282
pointer-cursor generic function 282
pointer-cursor-setter generic
    function 282
<pointer-drag-event> class 283
<pointer-enter-event> class 283
<pointer-event> class 284
<pointer-exit-event> class 284
<pointer-gesture> class 285
<pointer-motion-event> class 285
pointer-position generic function 286
pointer-sheet generic function 287
point-position generic function 42
points: init-keyword
    for <line> 77
    for <polygon> 87
    for <polyline> 90
    for <rectangle> 92
    for <reflection-underspecified> 45
    for <transform-underspecified> 62
point-x generic function 43
point-y generic function 43
<polygon> class 86
polygon? generic function 88
polygon-coordinates generic function 88
polygon-points generic function 89
<polyline> class 89
polyline? generic function 91
polyline-closed? generic function 91
popup-menu-callback: init-keyword
    for <list-control> 531
    for <table-control> 587
    for <tree-control> 598
<port> class 287
port generic function 288
port: init-keyword
    for <basic-user-pane> 400
    for <device-font> 130
    for <pointer> 279
    for <sheet> 299
    for <undefined-text-style-
        mapping> 324
port? generic function 289
port-modifier-state generic
    function 289
port-name generic function 290
port-pointer generic function 290
port-server-path generic function 291
<port-terminated-event> class 291
port-type generic function 292
<progress-bar> class 551
progress-bar: init-keyword
    for <status-bar> 573
progress-bar?: init-keyword
    for <status-bar> 573
<progress-note> class 738
*progress-note* variable 738
progress-note-label generic
    function 739
progress-note-label-setter generic
    function 739
<property-frame> class 739
<property-page> class 740
<push-box> class 552
<push-button> class 553
<push-menu-box> class 554
<push-menu-button> class 555

Q
queue-event generic function 292
queue-repaint generic function 293

R
<radio-box> class 556
<radio-button> class 557
<radio-menu-box> class 557
<radio-menu-button> class 558
radius-1-dx: init-keyword
    for <ellipse> 70
    for <elliptical-arc> 75
radius-1-dy: init-keyword
    for <ellipse> 70
    for <elliptical-arc> 75
radius-2-dx: init-keyword
    for <ellipse> 70
    for <elliptical-arc> 75
radius-2-dy: init-keyword
    for <ellipse> 70
    for <elliptical-arc> 75
raise-frame generic function 741
raise-progress-note generic
    function 742
raise-sheet generic function 293
ratios: init-keyword
    for <column-layout> 402
    for <row-layout> 424
read-image generic function 158
read-image-as generic function 159
read-only?: init-keyword
    for <gadget> 486
<rectangle> class 92
rectangle? generic function 93
rectangle-edges generic function 94

```

```

rectangle-height generic function 95
rectangle-max-point generic
    function 96
rectangle-max-position generic
    function 96
rectangle-min-point generic
    function 97
rectangle-min-position generic
    function 98
rectangle-size generic function 99
rectangle-width generic function 99
rectilinear-transform? generic
    function 44
$red constant 159
red: init-keyword
    for <color> 117
redo-command generic function 742
reflection-transform? generic
    function 44
<reflection-underspecified>
    class 45
<region> class 45
region: init-keyword
    for <basic-user-pane> 400
    for <sheet> 299
    for <window-event> 325
region? generic function 46
region-contains-position? generic
    function 46
region-contains-region? generic
    function 47
region-difference generic function 48
region-empty? generic function 48
region-equal generic function 49
region-intersection generic
    function 49
region-intersects-region? generic
    function 50
<region-set> class 50
region-set? generic function 51
region-set-function generic
    function 51
region-set-regions generic
    function 52
region-union generic function 53
relayout-children generic
    function 422
relayout-parent generic function 423
remove-child generic function 294
remove-colors generic function 160
remove-column generic function 559
remove-command generic function 743
remove-command-line-name generic
    function 743
remove-command-table function 743
remove-command-table-menu-item
    generic function 744
remove-item generic function 560
remove-node generic function 560
remove-presentation-translator
    generic function 744
repaint-sheet generic function 294
replace-child generic function 295
resizable?: init-keyword
    for <frame> 696
resizeable?: init-keyword
    for <layout> 414
resource-id: init-keyword
    for <frame> 696
restart-port generic function 295
restore-clipping-region generic
    function 392
<rich-text-editor> class 561
right: init-keyword
    for <bounding-box> 11
$right-button constant 295
rigid-transform? generic function 53
roots: init-keyword
    for <tree-control> 599
<row-layout> class 423
rows: init-keyword
    for <button-box> 473
    for <table-layout> 436
run-frame-top-level generic
    function 744

S
saturation: init-keyword
    for <color> 117
save-clipping-region generic
    function 392
scaling-transform? generic function 54
<scroll-bar> class 561
scroll-bars: init-keyword
    for <combo-box> 481
    for <list-box> 530
    for <list-control> 531
    for <table-control> 587
    for <text-editor> 592
    for <tree-control> 598
    for option-box> 549
<scroller> class 563
scrolling statement macro 563
scroll-position generic function 564

```

```

selection: init-keyword
  for <collection-gadget> 479
selection-mode: init-keyword
  for <collection-gadget> 479
<separator> class 564
set-box-edges generic function 54
set-box-position generic function 55
set-box-size generic function 55
set-caret-position generic function 296
set-frame-position generic function 744
set-frame-size generic function 745
set-pointer-position generic
  function 296
set-scroll-position generic
  function 565
set-sheet-edges generic function 297
set-sheet-position generic function 297
set-sheet-size generic function 298
<sheet> class 299
sheet: init-keyword
  for <caret> 189
  for <device-event> 214
  for <sheet-event> 307
sheet? generic function 303
sheet-ancestor? generic function 304
sheet-child generic function 304
sheet-children generic function 305
sheet-children-setter generic
  function 305
sheet-child-setter generic function 306
sheet-edges generic function 306
<sheet-event> class 307
sheet-event-mask generic function 308
sheet-event-mask-setter generic
  function 308
sheet-event-queue generic function 309
sheet-frame generic function 309
sheet-mapped? generic function 309
sheet-mapped?-setter generic
  function 310
sheet-medium generic function 310
sheet-parent generic function 311
sheet-parent-setter generic
  function 311
sheet-pointer-cursor generic
  function 312
sheet-pointer-cursor-setter generic
  function 312
sheet-position generic function 313
sheet-region generic function 313
sheet-region-setter generic
  function 314
sheet-size generic function 314
sheet-state generic function 315
sheet-text-cursor generic function 315
sheet-transform generic function 316
sheet-transform-setter generic
  function 317
sheet-viewport generic function 566
sheet-viewport-region generic
  function 567
sheet-withdrawn? generic function 317
$shift-key constant 317
show-buttons?: init-keyword
  for <tree-control> 598
show-edges?: init-keyword
  for <tree-control> 598
show-root-edges?: init-keyword
  for <tree-control> 598
<simple-command> class 745
<simple-frame> class 746
<simple-pane> class 426
<simple-status-bar> class 567
<simple-undoable-command> class 747
<single-child-composite-pane>
  class 427
<singular-transform> class 56
size: init-keyword
  for <text-style> 162
slant: init-keyword
  for <text-style> 162
<slider> class 567
slug-size: init-keyword
  for <scroll-bar> 561
$smallest-coordinate constant 56
$solid-pen constant 160
<space-requirement> class 427
space-requirement: init-keyword
  for <layout> 413
space-requirement? generic function 429
space-requirement-height generic
  function 430
space-requirement-max-height generic
  function 430
space-requirement-max-width generic
  function 431
space-requirement-min-height generic
  function 431
space-requirement-min-width generic
  function 432
space-requirement-width generic
  function 432
<spacing> class 569
spacing: init-keyword

```

```

        for <column-layout> 402
spacing:init-keyword
        for <row-layout> 424
<spin-box> class 570
split-bar-moved-callback: init-keyword
        for <splitter> 571
split-box-callback: init-keyword
        for <splitter> 571
<splitter> class 571, 573
splitter-split-bar-moved-callback
        generic function 572, 573
splitter-split-bar-moved-callback-setter generic function 573
<stack-layout> class 433
stack-layout-mapped-page generic
        function 434
stack-layout-mapped-page-setter generic function 435
start-angle: init-keyword
        for <ellipse> 70
        for <elliptical-arc> 75
start-dialog generic function 748
start-frame generic function 748
start-path generic function 392
start-x: init-keyword
        for <line> 77
start-y: init-keyword
        for <line> 77
state: init-keyword
        for <frame> 695
statement macro
        inhibit-updating-scroll-bars 528
statement macros
        horizontally 412
        labelling 529
        noting-progress 738
        scrolling 563
        tabling 440
        vertically 442
with-border 613
with-brush 326
with-clipping-region 328
with-cursor-visible 328
with-double-buffering 393
with-drawing-options 329
withdraw-sheet 330
with-frame-manager 331
with-identity-transform 331
with-output-to-pixmap 393
with-pen 332
with-pointer-grabbed 333
with-rotation 333
with-scaling 334
with-sheet-medium 335
with-spacing 613
with-text-style 336
with-transform 337
with-translation 337
<status-bar> class 573
status-bar: init-keyword
        for <simple-frame> 746, 747
status-bar-label-pane generic
        function 576
status-bar-label-pane-setter generic
        function 577
status-bar-progress-bar generic
        function 577
status-bar-progress-bar-setter generic function 577
status-code: init-keyword
        for <frame-exited-event> 704
<stencil> class 161
stencil? generic function 161
stipple: init-keyword
        for <brush> 112, 141
stretchable?: init-keyword
        for <pinboard-layout> 421
strikeout?: init-keyword
        for <text-style> 162
stroke-path generic function 352, 393
style-descriptor: init-keyword
        for <basic-user-pane> 400
        for <sheets> 299
substitute-numeric-argument-marker generic function 750
substitute-unsupplied-argument-marker generic function 750
$super-key constant 318
synchronize-display generic
        function 319

T
<tab-control> class 577
tab-control-current-page generic
        function 579
tab-control-current-page-setter generic function 580
tab-control-labels generic function 581

```

```

<tab-control-page> class 582
tab-control-pages generic function 583
tab-control-pages-setter generic
    function 584
<table-column> class 585
table-contents generic function 435
table-contents-setter generic
    function 435
<table-control> class 586
table-control-view generic function 591
<table-control-view> type 589
table-control-view-setter generic
    function 591
<table-item> class 592
<table-item-pane> class 592
<table-layout> class 436
tabling statement macro 440
test: init-keyword
    for <collection-gadget> 479
text: init-keyword
    for <text-gadget> 595
<text-cursor> class 592
<text-editor> class 592
<text-field> class 593
<text-gadget> class 595
text-size generic function 319
<text-style> class 162
text-style: init-keyword
    for <frame> 696
    for <sheet> 300
    for <undefined-text-style-
        mapping> 324
text-style? generic function 164
text-style-components generic
    function 164
text-style-family generic function 165
text-style-mapping generic function 321
text-style-mapping-exists? generic
    function 321
text-style-mapping-setter generic
    function 322
text-style-size generic function 166
text-style-slant generic function 167
text-style-strikeout? generic
    function 168
text-style-underline? generic
    function 168
text-style-weight generic function 169
thickness: init-keyword
    for <border> 469
    for <spacing> 569
thread: init-keyword
    for <frame> 695
tile: init-keyword
    for <brush> 112, 141
$tiles-stipple constant 170
<timer-event> class 323
timestamp: init-keyword
    for <event> 230
title: init-keyword
    for <frame> 695
<tool-bar> class 597
tool-bar: init-keyword
    for <simple-frame> 746, 747
top: init-keyword
    for <bounding-box> 11
<top-level-sheet> class 441
top-level-sheet generic function 323
top-level-sheet: init-keyword
    for <frame> 695
<transform> class 57
transform: init-keyword
    for <basic-user-pane> 400
    for <sheet> 299
    for <singular-transform> 56
    for <stencil> 161
transform? generic function 58
transform-angles generic function 58
transform-box generic function 59
transform-distance generic function 60
<transform-error> class 60
transform-position generic function 61
transform-region generic function 61
<transform-underspecified> class 62
translation-transform? generic
    function 62
<tree-control> class 598
tree-control-children-generator
    generic function 602
tree-control-children-generator-
    setter generic function 603
tree-control-children-predicate
    generic function 601
tree-control-children-predicate-
    setter generic function 602
tree-control-icon-function generic
    function 603
tree-control-initial-depth generic
    function 604
tree-control-initial-depth-setter
    generic function 605
tree-control-roots generic function 605
tree-control-roots-setter generic
    function 606

```

```

<tree-node> class 607
<tree-node-pane> class 608
ts-x: init-keyword
  for <brush> 112, 141
ts-y: init-keyword
  for <brush> 112, 141
tx: init-keyword
  for <transform> 57
ty: init-keyword
  for <transform> 57
type: init-keyword
  for <border> 469
  for <command-table-menu-item> 638
types
  <cursor> 211
  <list-control-view> 534
  <table-control-view> 589

```

## U

```

<undefined-text-style-mapping>
  class 324
underline?: init-keyword
  for <text-style> 162
undo-command generic function 750
undo-command: init-keyword
  for <simple-undoable-command> 747
units: init-keyword
  for <display> 215
  for <pen> 153
unsupplied-argument? function 750
$unsupplied-argument-marker
  constant 750
untransform-angles generic
  function 63
untransform-box generic function 63
untransform-distance generic
  function 64
untransform-position generic
  function 65
untransform-region generic
  function 65
update-callback: init-keyword
  for <menu> 541
  for <menu-bar> 542
  for <menu-box> 543
  for <menu-button> 544
  for <tool-bar> 597
update-gadget generic function 608
*user-command-table* variable 750

```

## V

```

value: init-keyword
  for <command-table-menu-item> 638
  for <status-bar> 573
  for <value-gadget> 609
value-changed-callback: init-keyword
  for <scroll-bar> 561
  for <value-gadget> 609
value-changing-callback: init-keyword
  for <scroll-bar> 561
  for <slider> 568
  for <text-gadget> 595
<value-gadget> class 608
value-key: init-keyword
  for <collection-gadget> 479
value-range: init-keyword
  for <status-bar> 573
  for value-range-gadget> 610
<value-range-gadget> class 609
value-type: init-keyword
  for <text-gadget> 595
variables
  *current-frame* 647
  *global-command-table* 725
  *progress-note* 738
  *user-command-table* 750
$vertical-hatch constant 170
vertically statement macro 442
vertical-scroll-bar: init-keyword
  for <viewport> 611
vertical-split-box?: init-keyword
  for <splitter> 572
view: init-keyword
  for <list-control> 531
  for <table-control> 586
<viewport> class 610
viewport? generic function 612
viewport-region generic function 612

```

## W

```

weight: init-keyword
  for <text-style> 162
$white constant 170
width: init-keyword
  for <caret> 189
  for <layout> 413
  for <pen> 153
  for <space-requirement> 427
  for <table-column> 585
widths: init-keyword
  for <table-control> 587

```

```
<window-configuration-event>
    class 324
<window-event> class 325
<window-repaint-event> class 326
with-border statement macro 613
with-brush statement macro 326
with-clipping-region statement
    macro 328
with-cursor-visible statement
    macro 328
with-double-buffering statement
    macro 393
with-drawing-options statement
    macro 329
withdrawn: init-keyword
    for <sheet> 299
withdraw-sheet thingy 330
with-frame-manager statement
    macro 331
with-identity-transform statement
    macro 331
with-output-to-pixmap statement
    macro 393
with-pen statement macro 332
with-pointer-grabbed statement
    macro 333
with-rotation statement macro 333
with-scaling statement macro 334
with-sheet-medium statement macro 335
with-spacing statement macro 613
with-text-style statement macro 336
with-transform statement macro 337
with-translation statement macro 337
<wizard-frame> class 751
<wizard-page> class 754
write-image generic function 171
```

## X

```
x: init-keyword
    for <caret> 189
    for <point> 41
    for <pointer-event> 284
    for <sheet> 299
x-alignment: init-keyword
    for <column-layout> 402
    for <table-layout> 437
    for <text-field> 594
$xor-brush constant 171
x-ratios: init-keyword
    for <table-layout> 436
x-ratios:init-keyword
    for <row-layout> 424
```

```
x-spacing: init-keyword
    for <row-layout> 424
    for <table-layout> 436
```

## Y

```
y: init-keyword
    for <caret> 189
    for <point> 41
    for <pointer-event> 284
    for <sheet> 299
y-alignment: init-keyword
    for <row-layout> 424
    for <table-layout> 437
$yellow constant 171
y-ratios: init-keyword
    for <column-layout> 402
    for <table-layout> 436
y-spacing: init-keyword
    for <column-layout> 402
    for <table-layout> 436
```

