8.1 Overview .......................................................... 253
8.2 The class hierarchy for DUIM-Layouts ................................. 253
8.3 DUIM-Layouts Module ............................................. 255

9 DUIM-Gadgets Library ........................................... 281
  9.1 Overview .......................................................... 281
  9.2 Callbacks and keys .............................................. 282
  9.3 Gadget protocols ............................................... 282
  9.4 The class hierarchy for DUIM-Gadgets .............................. 283
  9.5 Button gadgets ................................................ 288
  9.6 Text gadgets .................................................. 288
  9.7 Collection gadgets ............................................ 289
  9.8 Value range gadgets .......................................... 289
  9.9 Page gadgets .................................................. 290
  9.10 Gadgets that can have children ................................ 290
  9.11 DUIM-Gadgets Module ...................................... 291

10 DUIM-Frames Library ........................................ 387
  10.1 Overview .......................................................... 387
  10.2 The class hierarchy for DUIM-Frames .............................. 387
  10.3 DUIM-Commands Library ...................................... 389
  10.4 DUIM-Frames Module ........................................ 390

11 Indices and tables .............................................. 473

API Index .......................................................... 475

Index ............................................................... 487
Copyright © 1995-2000 Functional Objects, Inc.
Portions copyright © 2011-2016 Dylan Hackers.
Companies, names and data used in examples herein are fictitious unless otherwise noted.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Other brand or product names are the registered trademarks or trademarks of their respective holders.
CONVENTIONS IN THIS MANUAL

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

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.

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.

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 Library 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 Library** module provides more extensive support for co-ordinate geometry that is only required for more specialist uses.

The **DUIM-DCs Library** module provides color support to the DUIM library.

The **DUIM-Sheets Library** 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 Library** module provides support for graphics drawing

The **DUIM-Layouts Library** 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 Library** 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 Library** 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.

### 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*`.

### 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 take with volatile objects. For example, objects of class `<bounding-box>` are often volatile.
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.

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.

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
```
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.
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. DUIM-Geometry Module contains complete reference entries for each exposed interface.

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>.

The <region> class and its subclasses

The DUIM-Geometry library exposes the <region> class and its subclasses as shown in the following table. 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>
<thead>
<tr>
<th>&lt;region&gt;</th>
<th>&lt;region-set&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;point&gt;</td>
<td>&lt;path&gt;</td>
</tr>
<tr>
<td>&lt;area&gt;</td>
<td>&lt;bounding-box&gt;</td>
</tr>
</tbody>
</table>

• <region> This class is used to represent any set of points. The class: <region> class includes both bounded regions (that is, regions whose edges are known) and unbounded regions (that is, regions with no known edges).

• <region-set> This class represents a region set, that is, a set of regions.

• <point> This class is used to represent mathematical points (that is, regions with dimensionality 0).
• **<path>** The class **<path>** denotes bounded regions with a length, but no area (that is, they have dimensionality 1).
• **<area>** This class denotes bounded regions that have an area (that is, they have dimensionality 2).
• **<bounding-box>** A bounding box is an axis aligned rectangle that contains some region.

### 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 the following table. All the errors shown are subclasses of the **<error>** class. Note that the subclasses of **<transform-error>** are all specific to particular errors.

<table>
<thead>
<tr>
<th><strong>&lt;transform-error&gt;</strong></th>
<th><strong>&lt;transform-underspecified&gt;</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>&lt;reflection-underspecified&gt;</strong></td>
</tr>
<tr>
<td></td>
<td><strong>&lt;singular-transform&gt;</strong></td>
</tr>
</tbody>
</table>

• **<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.

### DUIM-Geometry Module

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

**= (<region>) Method**
Tests if its arguments are equal.

**Signature** = region1 region2 => boolean

**Parameters**
• **region1** – An instance of type **<region>**.
• **region2** – An instance of type **<region>**.

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

**Discussion** Tests if its arguments are equal. Returns #t if the two regions are the same, otherwise returns #f. Two regions are considered equal if they contain exactly the same set of points.

**= (<transform>) Method**
Tests if its arguments are equal.

**Signature** = transform1 transform2 => boolean

**Parameters**
• **transform1** – An instance of type **<transform>**.
• **transform2** – An instance of type **<transform>**.
Values

- **boolean** – An instance of type `<boolean>`.

**Discussion** Tests if its arguments are equal. Returns `#t` if the two transforms are the same, otherwise returns `#f`. Two transforms are considered equal if they transform every region the same way.

<area> Open Abstract Class
The class `<area>` denotes bounded regions that have dimensionality 2 (that is, have area).

**Superclasses** `<region>`

**Discussion**
The class `<area>` denotes bounded regions that have dimensionality 2 (that is, have area). `<area>` is a subclass of `<region>`.

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**

- **area?**

**See also**

- `<area>`

area? Generic function
Returns `#t` if its argument is an area, otherwise returns `#f`.

**Signature** `area? object => boolean`

**Parameters**

- **object** – An instance of type `<object>`.

**Values**

- **boolean** – An instance of type `<boolean>`.

**Discussion** Returns `#t` if `object` is an area, otherwise returns `#f`.

**See also**

- `<area>`

<bounding-box> Open Abstract Instantiable Class
The class that represents a bounding box.

**Superclasses** `<region>`

**Init-Keywords**

- **left** – An instance of type `<integer>`.
- **top** – An instance of type `<integer>`.
- **right** – An instance of type `<integer>`.
- **bottom** – An instance of type `<integer>`.

**Discussion**
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 rectilinear-transformation?.

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

• 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

See also

• bounding-box?
• bounding-box
• box-edges

bounding-box? Generic function

Returns true if its argument is a bounding box.

Signature  bounding-box? object => boolean

Parameters

• object – An instance of type <object>.

Values

• boolean – An instance of type <boolean>.

Discussion  Returns #t if object is a bounding box (that is, supports the bounding box protocol), otherwise returns #f.

See also

• <bounding-box>
• bounding-box
• box-edges
**bounding-box Generic function**

Returns the bounding box of a region.

**Signature**  
bounding-box region #key into => box

**Parameters**

- **region** – An instance of type `<region>`.
- **into** – An instance of type `false-or(<bounding-box>)`.

**Values**

- **box** – An instance of type `<bounding-box>`.

**Discussion**

The argument *region* 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.

This function often returns an existing object, so you should not modify the returned result.

If *into* is supplied, it is a bounding box that might be destructively modified to contain the result.

**See also**

- `<bounding-box>`
- `bounding-box?`
- `box-edges`

**box-bottom Function**

Returns the y coordinate of the bottom right corner of the bounding box of a region.

**Signature**  
box-bottom region => bottom

**Parameters**

- **region** – An instance of type `<region>`.

**Values**

- **bottom** – An instance of type `<integer>`.

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

**See also**

- `box-left`
- `box-right`
- `box-top`

**box-edges Generic function**

Returns the bounding box of a region.

**Signature**  
box-edges region => left top right bottom

**Parameters**

- **region** – An instance of type `<region>`.

**Values**

- **left** – An instance of type `<integer>`.
• **top** – An instance of type `<integer>`.
• **right** – An instance of type `<integer>`.
• **bottom** – An instance of type `<integer>`.

**Discussion**

Returns the bounding box of *region* as four integers specifying the \( x \) and \( y \) coordinates of the top left point and the \( x \) and \( y \) coordinates of the bottom right point of the box.

The argument *region* 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 *left*, *top*, *right*, and *bottom* will satisfy the inequalities:

\[
\text{left} \leq \text{right} \\
\text{top} \leq \text{bottom}
\]

**See also**

• `<bounding-box>`
• `bounding-box?`
• `bounding-box`

**box-height Function**

Returns the height of the bounding box of a region.

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

**Parameters**

• **region** – An instance of type `<region>`.

**Values**

• **height** – An instance of type `<integer>`.

**Discussion**  
Returns the height of the bounding box *region*. The height of a bounding box is the difference between the maximum \( y \) coordinate and its minimum \( y \) coordinate. The argument *region* must be either a bounded region or some other object that obeys the bounding box protocol.

**See also**

• `box-position`
• `box-size`
• `box-width`

**box-left Function**

Returns the \( x \) coordinate of the upper left corner of the bounding box of a region.

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

**Parameters**

• **region** – An instance of type `<region>`.

**Values**

• **left** – An instance of type `<integer>`.

**Discussion**  
Returns the \( x \) 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, such as a sheet.
See also
- \texttt{box-bottom}
- \texttt{box-right}
- \texttt{box-top}

\textbf{box-position} \textit{Generic function}
Returns the position of the bounding box of a region as two values.

\textbf{Signature} \hspace{1em} \texttt{box-position region \Rightarrow x \; y}

\textbf{Parameters}
- \texttt{region} – An instance of type \texttt{<region>}.

\textbf{Values}
- \texttt{x} – An instance of type \texttt{<integer>}.  
- \texttt{y} – An instance of type \texttt{<integer>}.  

\textbf{Discussion} Returns the position of the bounding box of \texttt{region} as two values. The position of a bounding box is specified by its top left point.

See also
- \texttt{box-height}
- \texttt{box-size}
- \texttt{box-width}

\textbf{box-right} \textit{Function}
Returns the \texttt{x} coordinate of the bottom right corner of the bounding box of a region.

\textbf{Signature} \hspace{1em} \texttt{box-right region \Rightarrow right}

\textbf{Parameters}
- \texttt{region} – An instance of type \texttt{<region>}.

\textbf{Values}
- \texttt{right} – An instance of type \texttt{<integer>}.  

\textbf{Discussion} Returns the \texttt{x} coordinate of the bottom right corner of the bounding box \texttt{region}. The argument \texttt{region} must be either a bounded region or some other object that obeys the bounding box protocol, such as a sheet.

See also
- \texttt{box-bottom}
- \texttt{box-left}
- \texttt{box-top}

\textbf{box-size} \textit{Generic function}
Returns the width and height of the bounding box of a region as two values

\textbf{Signature} \hspace{1em} \texttt{box-size region \Rightarrow width \; height}

\textbf{Parameters}
- \texttt{region} – An instance of type \texttt{<region>}.  

\textbf{Values}
• **width** – An instance of type `<integer>`.
• **height** – An instance of type `<integer>`.

**Discussion** Returns the width and height of the bounding box of `region` as two values. 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`
- `box-position`
- `box-width`

**box-top Function**

Returns the y coordinate of the upper left corner of the bounding box of a region.

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

**Parameters**

- **region** – An instance of type `<region>`.

**Values**

- **top** – An instance of type `<integer>`.

**Discussion** 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`
- `box-left`
- `box-right`

**box-width Function**

Returns the width of the bounding box of a region.

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

**Parameters**

- **region** – An instance of type `<region>`.

**Values**

- **width** – An instance of type `<integer>`.

**Discussion** 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`
- `box-position`
- `box-size`
**compose-rotation-with-transform** Generic function

Creates a new transform by composing a transform with the given rotation.

**Signature**

compose-rotation-with-transform transform angle #key origin => transform

**Parameters**

- **transform** – An instance of type `<transform>`.
- **angle** – An instance of type `<real>`.
- **origin** (#key) – An instance of type `<point>`. Default value: (0, 0).

**Values**

- **transform** – An instance of type `<transform>`.

**Discussion**

Creates a new transform by composing the transform `transform` with the given rotation. The order of composition is that the rotation transform is applied first, followed by the argument `transform`.

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

**See also**

- `make-rotation-transform`

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

Creates a new transform by composing a transform with the given scaling.

**Signature**

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

**Parameters**

- **transform** – An instance of type `<transform>`.
- **scale-x** – An instance of type `<real>`.
- **scale-y** – An instance of type `<real>`.
- **origin** (#key) – An instance of type `<point>`. Default value: (0, 0).

**Values**

- **transform** – An instance of type `<transform>`.

**Discussion**

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

- `make-scaling-transform`
compose-transforms Generic function
Returns a transform that is the mathematical composition of its arguments.

Signature  compose-transforms transform1 transform2 => transform

Parameters
• transform1 – An instance of type <transform>.
• transform2 – An instance of type <transform>.

Values
• transform – An instance of type <transform>.

Discussion
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 transform2 followed by the transform transform1.

See also
• compose-transform-with-rotation

compose-transform-with-rotation Generic function
Creates a new transform by composing a given rotation with a transform.

Signature  compose-transform-with-rotation transform angle #key origin => transform

Parameters
• transform – An instance of type <transform>.
• angle – An instance of type <real>.
• origin (#key) – An instance of type <point>. Default value: (0,0).

Values
• transform – An instance of type <transform>.

Discussion
Creates a new transform by composing a given rotation with the transform transform. The order of composition is transform first, followed by the rotation transform.

The argument angle represents the angle by which to rotate, in radians.

The argument origin represents the point about which to rotate. The default is to rotate around (0,0).

Note that this function could be implemented by using make-rotation-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
• make-rotation-transform

compose-transform-with-scaling Generic function
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

Parameters
• transform – An instance of type <transform>. 
• **scale-x** – An instance of type `<real>`.
• **scale-y** – An instance of type `<real>`.
• **origin (#key)** – An instance of type `<point>`. Default value: (0,0).

**Values**

• **transform** – An instance of type `<transform>`.

**Discussion**

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`
• `make-scaling-transform`

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

Creates a new transform by composing a given translation with a transform.

**Signature** `compose-transform-with-translation transform dx dy => transform`

**Parameters**

• **transform** – An instance of type `<transform>`.
• **dx** – An instance of type `<real>`.
• **dy** – An instance of type `<real>`.

**Values**

• **transform** – An instance of type `<transform>`.

**Discussion**

Creates a new transform by composing a given translation with the transform `transform`. The order of composition is `transform` first, followed by the translation transform.

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.

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

**See also**

• `make-translation-transform`
• `compose-transforms`
**compose-translation-with-transform** Generic function

Creates a new transform by composing a transform with the given translation.

**Signature**  
compose-translation-with-transform transform dx dy => transform

**Parameters**

- transform – An instance of type <transform>.
- dx – An instance of type <real>.
- dy – An instance of type <real>.

**Values**

- transform – An instance of type <transform>.

**Discussion**

Creates a new transform by composing the transform transform with the given translation. The order of composition is that the translation transform is applied first, followed by the argument transform.

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.

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

**See also**

- make-translation-transform
- compose-transforms

**do-coordinates** Function

Applies a function to each coordinate pair in its argument list.

**Signature**  
do-coordinates function coordinates => ()

**Parameters**

- function – An instance of type <function>.
- coordinates – An instance of type limited(<sequence>, of: <real>).

**Discussion**

Applies function to each coordinate pair in coordinates. The length of coordinates must be a multiple of 2. Function takes two arguments, the x and y value of each coordinate pair.

**do-endpoint-coordinates** Function

Applies a function to each coordinate pair in its argument list.

**Signature**  
do-endpoint-coordinates function coordinates => ()

**Parameters**

- function – An instance of type <function>.
- coordinates – An instance of type limited(<sequence>, of: <real>).

**Discussion**

Applies function to each pair of coordinate pairs in coordinates. The arguments coordinates 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, (x1, y1, x2, y2).

**do-regions** Generic function

Calls a function on each region in a set of regions.
**Signature**  do-regions function region #key normalize? => ()

**Parameters**
- **function** – An instance of type `<function>`.
- **region** – An instance of type `<region>`.
- **normalize? (#key)** – An instance of type `<boolean>`. Default value: #f.

**Discussion**  Calls function on each region in the region set region. This is often more efficient than calling region-set-regions. function is a function of one argument, a region. Region can be either a region set or a simple region, in which case function is called once on region itself. If normalize is supplied, it must be either #"x-banding" or #"y-banding". If it is #"x-banding" and all the regions in region are axis-aligned rectangles, the result is normalized by merging adjacent rectangles with banding done in the x direction. If it is #"y-banding" and all the regions in region 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 <region-set-not-rectangular> error.

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

Returns #t if the transform transform multiplies all x lengths and y lengths by the same magnitude, otherwise returns #f.

**Signature**  even-scaling-transform? transform => boolean

**Parameters**
- **transform** – An instance of type `<transform>`.

**Values**
- **boolean** – An instance of type `<boolean>`.

**Discussion**  Returns #t if the transform transform multiplies all x lengths and y lengths by the same magnitude, otherwise returns #f. even-scaling-transform? includes pure reflections through vertical and horizontal lines.

**$everywhere Constant**

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

**Type**  <region>

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

**See also**
- **$nowhere**

**fix-coordinate Function**

Coerces the given coordinate into an `<integer>`.

**Signature**  fix-coordinate coordinate => integer

**Parameters**
- **coordinate** – An instance of type `<real>`.

**Values**
- **integer** – An instance of type `<integer>`.

**Discussion**  Coerces the given coordinate into an `<integer>`.

**$identity-transform Constant**

An instance of a transform that is guaranteed to be an identity transform, that is, the transform that does nothing.
Type `<transform>`

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

See also
- `identity-transform?`

`identity-transform?` Generic function
Returns `#t` if a transform is equal (in the sense of `transform-equal`) to the identity transform.

Signature `identity-transform? transform => boolean`

Parameters
- `transform` – An instance of type `<transform>`.

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

Discussion Returns `#t` if the transform `transform` is equal (in the sense of `transform-equal`) to the identity transform, otherwise returns `#f`.

See also
- `$identity-transform`

`invert-transform` Generic function
Returns a transform that is the inverse of the given transform.

Signature `invert-transform transform => transform`

Parameters
- `transform` – An instance of type `<transform>`.

Values
- `transform` – An instance of type `<transform>`.

Conditions
If `transform` is singular, `invert-transform` signals the `<singular-transform>` error.

Note: With finite-precision arithmetic there are several low-level conditions that might occur during the attempt to invert a singular or almost singular transform. (These include computation of a zero determinant, floating-point underflow during computation of the determinant, or floating-point overflow during subsequent multiplication.) `invert-transform` signals the `<singular-transform>` error for all of these cases.

Discussion Returns a transform that is the inverse of the transform `transform`. The result of composing a transform with its inverse is equal to the identity transform.

See also
- `invertible-transform?`

`invertible-transform?` Generic function
Returns `#t` if the given transform has an inverse.

Signature `invertible-transform? transform => boolean`

Parameters
• **transform** – An instance of type `<transform>`.

**Values**

• **boolean** – An instance of type `<boolean>`.

**Discussion** Returns `#t` if the transform `transform` has an inverse, otherwise returns `#f`.

**See also**

• `invert-transform`

$largest-coordinate Constant

The largest valid coordinate.

**Type** `<integer>`

**Discussion** The largest valid coordinate.

**See also**

• `$smallest-coordinate`

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

Returns a transform that takes points `point-1` into `point-1-image`, `point-2` into `point-2-image` and `point-3` into `point-3-image`.

**Signature**

```
make-3-point-transform x1 y1 x2 y2 x3 x1-image y1-image x2-image y2-image x3-image => transform
```

**Signature**

```
make-3-point-transform* point-1 point-2 point-3 point-1-image point-2-image point-3-image => transform
```

The following arguments are specific to `make-3-point-transform`.

**Parameters**

• `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>`.
• `x1-image` – An instance of type `<real>`.
• `y1-image` – An instance of type `<real>`.
• `x2-image` – An instance of type `<real>`.
• `y2-image` – An instance of type `<real>`.
• `x3-image` – An instance of type `<real>`.
• `y3-image` – An instance of type `<real>`.

The following arguments are specific to `make-3-point-transform*`.

**Parameters**

• `point-1` – An instance of type `<point>`.
• `point-2` – An instance of type `<point>`.
• `point-3` – An instance of type `<point>`.
• **point-1-image** – An instance of type `<point>`.
• **point-2-image** – An instance of type `<point>`.
• **point-3-image** – An instance of type `<point>`.

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

Conditions
If `point-1`, `point-2` and `point-3` are colinear, the `<transform-underspecified>` error is signalled. If `point-1-image`, `point-2-image` and `point-3-image` are colinear, the resulting transform will be singular (that is, will have no inverse) but this is not an error.

Discussion
Returns a transform that takes points `point-1` into `point-1-image`, `point-2` into `point-2-image` and `point-3` into `point-3-image`. Three non-colinear points and their images under the transform are enough to specify any affine transformation.

The function `make-3-point-transform*` is identical to `make-3-point-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.

### make-bounding-box Function

Returns an object of the class `<bounding-box>`.

**Signature**

```
make-bounding-box x1 y1 x2 y2 => box
```

**Parameters**

• **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**

• **box** – An instance of type `<bounding-box>`.

**Discussion**

Returns an object of the class `<bounding-box>` with the edges specified by `x1`, `y1`, `x2`, and `y2`. `x1`, `y1`, `x2`, and `y2` are canonicalized in the following way. The min point of the box has an x coordinate that is the smaller of `x1` and `x2` and a y coordinate that is the smaller of `y1` and `y2`. The max point of the box has an x coordinate that is the larger of `x1` and `x2` and a y coordinate that is the larger of `y1` and `y2`. (Therefore, in a right-handed coordinate system the canonicalized values of `x1`, `y1`, `x2`, and `y2` correspond to the left, top, right, and bottom edges of the box, respectively.)

This is a convenient shorthand function for `make(<bounding-box>, left: top: right: bottom:)`.

### make-point Function

Returns an object of class `<point>`.

**Signature**

```
make-point x y => point
```

**Parameters**

• **x** – An instance of `<real>`.
• **y** – An instance of `<real>`.

**Values**


• **point** – An instance of type `<point>`.

**Discussion** Returns an object of class `<point>` whose coordinates are x and y.

**make-reflection-transform Function**

Returns a transform that reflects every point through the line passing through the positions x1,y1 and x2,y2.

**Signature** make-reflection-transform x1 y1 x2 y2 => transform

**Parameters**

• 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**

• transform – An instance of type `<transform>`. The resultant transformation.

**Discussion**

Returns a transform that reflects every point through the line passing through the positions x1,y1 and x2,y2.

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.

**See also**

• **make-rotation-transform**
• **make-scaling-transform**
• **make-transform**
• **make-translation-transform**
• `<reflection-underspecified>`

**make-reflection-transform* Function**

Returns a transform that reflects every point through the line passing through the points point1 and point2.

**Signature** make-reflection-transform* point-1 point-2 => transform

**Parameters**

• point1 – An instance of type `<point>`. The first point.
• point2 – An instance of type `<point>`. The second point.

**Values**

• transform – An instance of type `<transform>`. The resultant transformation.

**Discussion**

Returns a transform that reflects every point through the line passing through the points point1 and point2.
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 `:func: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
• `make-rotation-transform`
• `make-scaling-transform`
• `make-transform`
• `make-translation-transform`
• `<reflection-underspecified>`

### make-rotation-transform Function

Returns a transform that rotates all points by `angle` around the point specified by coordinates `origin-x` and `origin-y` or the point object `origin`.

**Signature**

`make-rotation-transform angle #key origin-x origin-y => transform`

**Signature**

`make-rotation-transform* angle #key origin => transform`

**Parameters**

• `angle` – An instance of type `<real>`.

The following arguments are specific to `make-rotation-transform`.

**Parameters**

• `origin-x` – An instance of type `<real>`. Default value: 0.
• `origin-y` – An instance of type `<real>`. Default value: 0.

The following argument is specific to `make-reflection-transform*`.

**Parameters**

• `origin` – An instance of type `<point>`. Default value: (0, 0).

**Values**

• `transform` – An instance of type `<transform>`.

**Discussion**

Returns a transform that rotates all points by `angle` around the point specified by coordinates `origin-x` and `origin-y` or the point object `origin`. The angle must be expressed in radians.

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.

The function `make-rotation-transform*` is identical to `make-rotation-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`
• `make-scaling-transform`
**make-scaling-transform** Function

Returns a transform that multiplies the x-coordinate distance of every point from `origin` by `scale-x` and the y-coordinate distance of every point from `origin` by `scale-y`.

**Signature**

```
make-scaling-transform scale-x scale-y #key origin-x origin-y => transform
```

**Signature**

```
make-scaling-transform* scale-x scale-y #key origin => transform
```

**Parameters**

- **scale-x** – An instance of type `<real>`.
- **scale-y** – An instance of type `<real>`.

The following arguments are specific to `make-scaling-transform`.

**Parameters**

- **origin-x** – An instance of type `<real>`. Default value: 0.
- **origin-y** – An instance of type `<real>`. Default value: 0.

The following argument is specific to `make-scaling-transform*`.

**Parameters**

- **origin** – An instance of type `<point>`.

**Values**

- **transform** – An instance of type `<transform>`. The resultant transformation.

**Discussion**

Returns a transform that multiplies the x-coordinate distance of every point from `origin` by `scale-x` and the y-coordinate distance of every point from `origin` by `scale-y`.

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 arguments `origin-x` and `origin-y` represent the point around which scaling is performed. The default is to scale around the origin.

There is no single definition of a scaling transformation. Transforms that preserve all angles and multiply all lengths by the same factor (preserving the shape of all entities) are certainly scaling transformations. However, scaling is also used 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`
- `make-rotation-transform`
- `make-transform`
- `make-translation-transform`
**make-transform Function**

Returns a general affine transform.

**Signature**

```
make-transform mxx mxy myx myy tx ty => transform
```

**Parameters**

- `mxx` – An instance of type `<real>`.
- `mxy` – An instance of type `<real>`.
- `myx` – An instance of type `<real>`.
- `myy` – An instance of type `<real>`.
- `tx` – An instance of type `<real>`.
- `ty` – An instance of type `<real>`.

**Values**

- `transform` – An instance of type `<transform>`.

**Discussion**

Returns a general transform whose effect is:

\[
x' = mxx \cdot x + mxy \cdot y + tx
\]

\[
y' = myx \cdot x + myy \cdot y + ty
\]

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**

- `make-reflection-transform`
- `make-rotation-transform`
- `make-scaling-transform`
- `make-translation-transform`

**make-translation-transform Function**

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
```

**Parameters**

- `dx` – An instance of type `<real>`.
- `dy` – An instance of type `<real>`.

**Values**

- `transform` – An instance of type `<transform>`.

**Discussion**

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`
- `make-rotation-transform`
- `make-scaling-transform`
- `make-transform`

$\text{nowhere}$ Constant

The empty region, the opposite of $\text{everywhere}$.

Type `<region>`

Discussion The empty region, the opposite of $\text{everywhere}$.

See also

- `$\text{everywhere}$`

`<path>` Open Abstract Class

The class `<path>` denotes bounded regions that have dimensionality 1 (that is, have length).

Superclasses `<region>`

Discussion

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 $\text{nowhere}$.

Operations

- `<path>`

See also

- `<path>`

`path?` Generic function

Returns #t if its argument is a path.

Signature `path? object => boolean`

Parameters

- `object` – An instance of type `<object>`.

Values

- `boolean` – An instance of type `<boolean>`.

Discussion Returns #t if `object` is a path, otherwise returns #f.

See also

- `<path>`

`<point>` Open Abstract Instantiable Class

The class that corresponds to a mathematical point.

Superclasses `<region>`

Init-Keywords
• x – An instance of type <integer>.
• y – An instance of type <integer>.

Discussion 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

• =
• 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
Returns true if object is a point.

Signature point? object => boolean

Parameters

• object – An instance of type <object>.

Values

• boolean – An instance of type <boolean>.

Discussion Returns #t if object is a point.

point-position Generic function
Returns both the x and y coordinates of a point.

Signature point-position point => x y

Parameters

• point – An instance of type <point>.

Values

• x – An instance of type <real>.
• y – An instance of type <real>.

Discussion Returns both the x and y coordinates of the point point as two values.

See also

• point-x
• point-y
point-x Generic function
Returns the $x$ coordinate of a point.

Signature point-x point => x

Parameters
• point – An instance of type <point>.

Values
• $x$ – An instance of type <real>.

Discussion Returns the $x$ coordinate of point.

See also
• point-position
• point-y

point-y Generic function
Returns the $y$ coordinate of a point.

Signature point-y point => y

Parameters
• point – An instance of type <point>.

Values
• $y$ – An instance of type <real>

Discussion Returns the $y$ coordinate of point.

See also
• point-position
• point-x

rectilinear-transform? Generic function
Returns #t if a transform always transforms any axis-aligned rectangle into another axis-aligned rectangle.

Signature rectilinear-transform? transform => boolean

Parameters
• transform – An instance of type <transform>.

Values
• boolean – An instance of type <boolean>.

Discussion
Returns #t if the transform $\text{transform}$ always transforms any axis-aligned rectangle into another axis-aligned rectangle, otherwise returns #f.

This category includes scalings as a subset, and also includes 90 degree rotations.

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.

reflection-transform? Generic function
Returns #t if the transform inverts the handedness of the coordinate system.
Signature reflection-transform? transform => boolean

Parameters

• transform – An instance of type <transform>.

Values

• boolean – An instance of type <boolean>.

Discussion

Returns #t if the transform transform inverts the handedness 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> Concrete Sealed Class

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

Superclasses <transform-underspecified>

Init-Keywords

• points – Instances of type <point>.

Discussion The error that is signalled when make-reflection-transform is given two coincident points. This condition handles the points: initarg, which is used to supply the points that are in error.

See also

• make-reflection-transform

<region> Open Abstract Class

The class that corresponds to a set of points.

Superclasses <object>

Discussion

The class that corresponds to a set of points. The class: <region> 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

• =

• 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

See also
• region?

region? Generic function
Returns #t if its argument is a region.

Signature  region? object => boolean
Parameters
• object – An instance of type <object>.

Values
• boolean – An instance of type <boolean>.

Discussion  Returns #t if object is a region, otherwise returns ‘#f’.
See also
• <region>

region-contains-position? Generic function
Returns #t if the point at x,y is contained in the region.

Signature  region-contains-position? region x y => boolean
Parameters
• region – An instance of type <region>.
• x – An instance of type <real>.
• y – An instance of type <real>.

Values
• boolean – An instance of type <boolean>.

Discussion  Returns #t if the point at x,y is contained in the region region, otherwise returns #f. Since regions in DUIM are closed, this returns #t if the point at x,y is on the region’s boundary.
See also
• region-contains-region?

region-contains-region? Generic function
Returns #t if all points in the second region are members of the first region.

Signature  region-contains-region? region1 region2 => boolean
Parameters
• region1 – An instance of type <region>.
• region2 – An instance of type <region>.

Values
• boolean – An instance of type <boolean>.
Discussion Returns #t if all points in the region region2 are members of the region region1, otherwise returns #f. region-contains-position? is a special case of region-contains-region? in which the region is the point x,y.

See also
• region-contains-position?

region-difference Generic function
Returns a region that contains all points in the region region1 that are not in the region region2 (possibly plus additional boundary points to make the result closed).

Signature region-difference region1 region2 => region

Parameters
• region1 – An instance of type <region>.
• region2 – An instance of type <region>.

Values
• region – An instance of type <region>.

Discussion
Returns a region that contains all points in the region region1 that are not in the region region2 (possibly plus additional boundary points to make the result closed).

The result of region-difference has the same dimensionality as region1, or is $nowhere. 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.

Note: region-difference may return either a simple region or a region set.

region-empty? Generic function
Returns #t if the region is empty.

Signature region-empty? region => boolean

Parameters
• region – An instance of type <region>.

Values
• boolean – An instance of type <boolean>.

Discussion Returns #t if the region is empty, otherwise returns #f.

region-equal Generic function
Returns #t if the two regions region1 and region2 contain exactly the same set of points.

Signature region-equal region1 region2 => boolean

Parameters
• region1 – An instance of type <region>.
• region2 – An instance of type <region>.

Values
• boolean – An instance of type <boolean>.
Discussion Returns \( \#t \) if the two regions \( \text{region1} \) and \( \text{region2} \) contain exactly the same set of points, otherwise returns \( \#f \). There is a method on \( = \) on \(<\text{region}>\) and \(<\text{region}>\) that calls \text{region-equal}.

\text{region-intersection} Generic function

Returns the intersection of two regions, as a region.

Signature \( \text{region-intersection} \text{ region1 region2 } => \text{ region} \)

Parameters

• \( \text{region1} \) – An instance of type \(<\text{region}>\).

• \( \text{region2} \) – An instance of type \(<\text{region}>\).

Values

• \( \text{region} \) – An instance of type \(<\text{region}>\).

Discussion

Returns a region that contains all points that are in both of the regions \( \text{region1} \) and \( \text{region2} \) (possibly with some points removed in order to satisfy the dimensionality rule).

The result of \text{region-intersection} has dimensionality that is the minimum dimensionality of \( \text{region1} \) and \( \text{region2} \), or is \$nowhere\$. For example, the intersection of two areas is either another area or \$nowhere\$; 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: \text{region-intersection} may return either a simple region or a region set.

See also

• \( \text{region-union} \)

\text{region-intersects-region?} Generic function

Returns \( \#f \) if two regions do not intersect**.

Signature \( \text{region-intersects-region?} \text{ region1 region2 } => \text{ boolean} \)

Parameters

• \( \text{region1} \) – An instance of type \(<\text{region}>\).

• \( \text{region2} \) – An instance of type \(<\text{region}>\).

Values

• \( \text{boolean} \) – An instance of type \(<\text{boolean}>\).

Discussion Returns \( \#f \) if \text{region-intersection} of the two regions \( \text{region1} \) and \( \text{region2} \) would be \$nowhere\$ (that is, they do not intersect), otherwise returns \#t.

\(<\text{region-set}>\) Open Abstract Class

The class that represents a region set.

Superclasses \(<\text{region}>\)

Discussion The class that represents a region set; a subclass of \(<\text{region}>\).

Operations

• \( \text{box-edges} \)

• \( \text{do-regions} \)
• region-contains-position?
• region-contains-region?
• region-difference
• region-empty?
• region-intersection
• region-set-function
• region-set-regions
• region-union
• transform-region

See also
• region-set?

region-set? Generic function
Returns #t if its argument is a region set.

Signature  region-set? object => boolean
Parameters
• object – An instance of type <object>.
Values
• boolean – An instance of type <boolean>.
Discussion Returns #t if object is a region set, otherwise returns #f.
See also
• <region-set>

region-set-function Generic function
Returns the function that composed the region.

Signature  region-set-function region => function
Parameters
• region – An instance of type <region>.
Values
• function – An instance of type <function>.
Discussion Returns the function that composed the region, region-intersection, region-union, or region-difference.

region-set-regions Generic function
Returns a sequence of the regions in the region set.

Signature  region-set-regions region #key normalize? => regions
Parameters
• region – An instance of type <region>.
• normalize? – one-of(#f, "x-banding", "y-banding"). Default value: #f.
Values

36 Chapter 3. DUIM-Geometry Library
regions – An instance of type limited(sequence), of: <region>.

Conditions Normalizing a region set that is not composed entirely of axis-aligned rectangles using x- or y-banding causes DUIM to signal the <region-set-not-rectangular> error.

Discussion

Returns a sequence of the regions in the region set region. region can be either a region set or a simple region, in which case the result is simply a sequence of one element: region.

For the case of region sets that are unions of axis-aligned rectangles, the rectangles returned by region-set-regions are guaranteed not to overlap. If normalize? is supplied, it must be either #"x-banding" or #"y-banding". If it is #"x-banding" and all the regions in region are axis-aligned rectangles, the result is normalized by merging adjacent rectangles with banding done in the x direction. If it is #"y-banding" and all the regions in region are rectangles, the result is normalized with banding done in the y direction.

region-union Generic function

Returns the union of two regions, as a region.

Signature  region-union region1 region2 => region

Parameters

• region1 – An instance of type <region>.
• region2 – An instance of type <region>.

Values

• region – An instance of type <region>.

Discussion

Returns a region that contains all points that are in either of the regions region1 or region2 (possibly with some points removed in order to satisfy the dimensionality rule)

The result of region-union always has dimensionality that is the maximum dimensionality of region1 and region2. For example, the union of a path and an area produces an area; the union of two paths is a path.

Note: region-union may return either a simple region or a region set.

See also

• region-intersection

rigid-transform? Generic function

Returns #t if the transform transforms the coordinate system as a rigid object.

Signature  rigid-transform? transform => boolean

Parameters

• transform – An instance of type <transform>.

Values

• boolean – An instance of type <boolean>.

Discussion

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

**scaling-transform? Generic function**

Returns #t if the transform `transform` multiplies all x lengths by one magnitude and all y lengths by another magnitude, otherwise returns #f.

**Signature** `scaling-transform? transform ==> boolean`

**Parameters**

- `transform` – An instance of type `<transform>`.

**Values**

- `boolean` – An instance of type `<boolean>`.

**Discussion** Returns #t if the transform `transform` multiplies all x lengths by one magnitude and all y lengths by another magnitude, otherwise returns #f. This category includes even scalings as a subset.

**set-box-edges Generic function**

Sets the edges of a box and returns the bounding box.

**Signature** `set-box-edges box left top right bottom ==> box`

**Parameters**

- `box` – An instance of type `<bounding-box>`.
- `left` – An instance of type `<integer>`.
- `top` – An instance of type `<integer>`.
- `right` – An instance of type `<integer>`.
- `bottom` – An instance of type `<integer>`.

**Values**

- `box` – An instance of type `<bounding-box>`.

**Discussion** Sets the edges of a box and returns the bounding box `box`. This might destructively modify `box` or it might not, depending on what class `box` is.

**set-box-position Generic function**

Sets the position of the bounding box and returns a (possibly new) box.

**Signature** `set-box-position box x y ==> box`

**Parameters**

- `box` – An instance of type `<bounding-box>`.
- `x` – An instance of type `<real>`.
- `y` – An instance of type `<real>`.

**Values**

- `box` – An instance of type `<bounding-box>`.

**Discussion** Sets the position of the bounding box `box` and might or might not modify the box.

**set-box-size Generic function**

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

**Signature** `set-box-size box width height ==> box`
Parameters

- **box** – An instance of type `<bounding-box>`.
- **width** – An instance of type `<integer>`.
- **height** – An instance of type `<integer>`.

Values

- **box** – An instance of type `<bounding-box>`.

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

**<singular-transform>** Instantiable Sealed Class

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.

Discussion
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.

See also
- `invert-transform`

**$smallest-coordinate** Constant

The smallest valid coordinate.

**Type** `<integer>`

**Discussion** The smallest valid coordinate. Coordinates must be instances of type `<integer>`.

See also
- `$largest-coordinate`

**<transform>** Open Abstract Instantiable Class

The superclass of all transforms.

**Superclasses** `<object>`

**Init-Keywords**
- **mxx** – An instance of type `<real>`.
- **mxy** – An instance of type `<real>`.
- **myx** – An instance of type `<real>`.
- **myy** – An instance of type `<real>`.
- **tx** – An instance of type `<real>`.
- **ty** – An instance of type `<real>`.
Discussion  The superclass of all transforms. There are one or more subclasses of \(<transform>\) 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

- \(=\)
- \(\text{compose-rotation-with-transform}\)
- \(\text{compose-scaling-with-transform}\)
- \(\text{compose-transforms}\)
- \(\text{compose-transform-with-translation}\)
- \(\text{compose-translation-with-transform}\)
- \(\text{even-scaling-transform}\)
- \(\text{identity-transform}\)
- \(\text{invert-transform}\)
- \(\text{invertible-transform}\)
- \(\text{rectilinear-transform}\)
- \(\text{reflection-transform}\)
- \(\text{rigid-transform}\)
- \(\text{scaling-transform}\)
- \(\text{transform-angles}\)
- \(\text{transform-box}\)
- \(\text{transform-distance}\)
- \(\text{transform-position}\)
- \(\text{transform-region}\)
- \(\text{translation-transform}\)
- \(\text{untransform-angles}\)
- \(\text{untransform-box}\)
- \(\text{untransform-distance}\)
- \(\text{untransform-position}\)
- \(\text{untransform-region}\)

See also

- \(\text{transform}\)

\text{transform}?  Generic function

Returns \(\texttt{#t}\) if its argument is a transform.

Signature  \(\text{transform? \ object} => \text{boolean}\)

Parameters

- \(\text{object}\) – An instance of type \(<\text{object}>\).
Values

- boolean – An instance of type `<boolean>`.

Discussion Returns `#t` if `object` is a transform, otherwise returns `#f`.

See also

- `<transform>`

**transform-angles** Generic function

Applies the transform to the start and end angles of an object, and returns the transformed angles.

**Signature** transform-angles transform start-angle end-angle => new-start new-end

**Parameters**

- transform – An instance of type `<transform>`.
- start-angle – An instance of type `<real>`.
- end-angle – An instance of type `<real>`.

**Values**

- new-start – An instance of type `<real>`.
- new-end – An instance of type `<real>`.

Discussion Applies the transform `transform` to the angles `start-angle` and `end-angle` of an object, and returns the transformed angles.

**transform-box** Generic function

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

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

**Parameters**

- 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>`.

Discussion 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?`.
transform-distance Generic function
Applies a transform to a distance represented by the coordinate arguments and returns the transformed coordinates.

Signature transform-distance transform dx dy => dx dy

Parameters

  • transform – An instance of type <transform>.
  • dx – An instance of type <real>.
  • dy – An instance of type <real>.

Values

  • dx – An instance of type <real>.
  • dy – An instance of type <real>.

Discussion Applies the transform transform to the distance represented by dx and dy, and returns the transformed dx and dy. A distance represents the difference between two points. It does not transform like a point.

<transform-error> Sealed Class
The superclass of all error conditions distributed when there is an error with a transform.

Superclasses <error>

Discussion The class that is the superclass of three error conditions, <transform-underspecified>, <reflection-underspecified>, and <singular-transform>.

transform-position Generic function
Applies a transform to the point whose coordinates are x and y.

Signature transform-position transform x y => new-x new-y

Parameters

  • transform – An instance of type <transform>.
  • x – An instance of type <real>
  • y – An instance of type <real>

Values

  • new-x – An instance of type <real>
  • new-y – An instance of type <real>

Discussion Applies the transform transform to the point whose coordinates are x and y. transform-position is the spread version of transform-region in the case where the region is a point.

transform-region Generic function
Applies a transform to a region, and returns the transformed region.

Signature transform-region transform region => region

Parameters

  • transform – An instance of type <transform>.
  • region – An instance of type <region>.

Values
• **region** – An instance of type `<region>`.

**Discussion** Applies `transform` to the region `region`, and returns the transformed region.

**<transform-underspecified> Concrete Sealed Class**

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

**Superclasses** `<transform-error>`

**Init-Keywords**

• **points** – The points that are in error.

**Discussion** The error that is signalled when `make-3-point-transform` is given three colinear image points. This condition handles the `points`: initarg, which is used to supply the points that are in error.

**See also**

• `make-3-point-transform`

**translation-transform? Generic function**

Returns `#t` if a transform is a pure translation, that is, a transform such that there are two distance components transform `dx` and `dy` and every point `(x,y)` is moved to `(x+dx,y+dy)`.

**Signature** `translation-transform? transform => boolean`

**Parameters**

• `transform` – An instance of type `<transform>`.

**Values**

• `boolean` – An instance of type `<boolean>`.

**Discussion** Returns `#t` if the transform `transform` is a pure translation, that is, a transform such that there are two distance components transform `dx` and `dy` and every point `(x,y)` is moved to `(x+dx,y+dy)`. Otherwise, `translation-transform?` returns `#f`.

**untransform-angles** Generic function

Undoes a transform and returns the original start and end angles of the object.

**Signature** `untransform-angles transform start-angle end-angle => orig-start orig-end`

**Parameters**

• `transform` – An instance of type `<transform>`.
• `start-angle` – An instance of type `<real>`.
• `end-angle` – An instance of type `<real>`.

**Values**

• `orig-start` – An instance of type `<real>`.
• `orig-end` – An instance of type `<real>`.

**Conditions**

• `<singular-transform>` cannot be inverted.

**Discussion**

Undoes the transform `transform` to the angles `new-start` and `new-end`, returning the original `orig-start` and `orig-end`. This is exactly equivalent to:
transform-box Generic function
Undoes the previous transformation on the rectangle left, top and right, bottom, returning the original box.

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

Parameters
• 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>.

Conditions
• <singular-transform> cannot be inverted.

Discussion
Undoes the previous transformation on the rectangle top-left-x, top-left-y and bottom-right-x, bottom-right-y, returning the original box. This is exactly equivalent to:

\[
\text{transform-box(invert-transform(*transform*))}
\]

untransform-distance Generic function
Undoes the previous transformation on the distance dx,dy, returning the original dx,dy.

Signature  untransform-distance transform dx dy => dx dy

Parameters
• transform – An instance of type <transform>.
• dx – An instance of type <real>.
• dy – An instance of type <real>.

Values
• dx – An instance of type <real>.
• dy – An instance of type <real>.

Conditions
• <singular-transform> cannot be inverted.

Discussion
Undoes the previous transformation on the distance dx,dy, returning the original dx,dy. This is exactly equivalent to:

\[
\text{transform-box(invert-transform(*transform*))}
\]
untransform-position Generic function
Undoes the previous transformation on the point x, y, returning the original point.

Signature untransform-position transform x y => x y

Conditions
- <singular-transform> cannot be inverted.

Discussion
Undoes the previous transformation on the point x, y, returning the original point. This is exactly equivalent to:

```
transform-position(invert-transform(*transform*))
```

untransform-region Generic function
Undoes the previous transformation on a region, returning the original region.

Signature untransform-region transform region2 => region1
Parameters
- transform – An instance of type <transform>.
- region2 – An instance of type <region>. The region to untransform.

Values
- region1 – An instance of type <region>. The original region.

Conditions
- <singular-transform> cannot be inverted.

Discussion
Undoes the previous transformation on the region region, returning the original region. This is exactly equivalent to

```
transform-region(invert-transform(*transform region*)))
```
DUIM-EXTENDED-GEOMETRY LIBRARY

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. ‘See DUIM-Extended-Geometry Module’ contains complete reference entries for each exposed interface.

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.

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.

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.
DUIM-Extended-Geometry Module

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

do-polygon-coordinates Generic function
Applies a function to all of the coordinates of the vertices of a polygon.

Signature  do-polygon-coordinates function polygon => ()

Parameters
•  function – An instance of type <function>.
•  polygon – An instance of type <polygon>.

Discussion  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

do-polygon-segments Generic function
Applies a function to the segments that compose a polygon.

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

Parameters
•  function – An instance of type <function>.
•  polygon – An instance of type <polygon>.

Discussion  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

draw-design Generic function
Draws a design on a drawing surface.

Signature  draw-design drawable design => ()

Parameters
•  drawable – An instance of type type-union(<sheet>, <medium>).
•  design – A <region> to draw.

Discussion  Draws design on the sheet medium drawable.

<ellipse> Abstract Instantiable Class
The class that corresponds to an ellipse.

Superclasses  <area>
Init-Keywords
• center-x – An instance of type <real>.
• center-y – An instance of type <real>.
• center-point – An instance of type <point>.
• radius-1-dx – An instance of type <real>
• radius-1-dy – An instance of type <real>
• radius-2-dx – An instance of type <real>
• radius-2-dy – An instance of type <real>
• start-angle – An instance of false-or(<real>).
• end-angle – An instance of false-or(<real>).

Discussion

An ellipse 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: \( y^c - dx*1 + dx*2 \)

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>
• make-ellipse

ellipse? Generic function

Returns #t if an object is an ellipse.
**Signature**  
`ellipse? object => boolean`

**Parameters**
- `object` – An instance of type `<object>`.

**Values**
- `boolean` – An instance of type `<boolean>`.

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

**See also**
- `<ellipse>`

**ellipse-center-point**  
Generic function

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

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

**Parameters**
- `elliptical-object` – An instance of type `type-union(<ellipse>, <elliptical-arc>)`.

**Values**
- `point` – An instance of type `<point>`.

**Discussion**  
Returns the center point of `ellipse-object` as a `<point>` object.

**See also**
- `make-ellipse`

**ellipse-center-position**  
Generic function

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

**Signature**  
`ellipse-center-position* elliptical-object => x y`

**Parameters**
- `elliptical-object` – An instance of type `type-union(<ellipse>, <elliptical-arc>)`.

**Values**
- `x` – An instance of type `<real>`.
- `y` – An instance of type `<real>`.

**Discussion**

Returns the coordinates of the center point of `elliptical-object`.

The arguments `x` and `y` represent the x and y coordinates of the center of the elliptical object, respectively.

**See also**
- `make-ellipse`

**ellipse-end-angle**  
Generic function

Returns the end angle of an ellipse or an elliptical-object.

**Signature**  
`ellipse-end-angle elliptical-object => angle`

**Parameters**
• **elliptical-object** – An instance of type `type-union(<ellipse>, <elliptical-arc>)`.

**Values**

• **angle** – An instance of type `false-or(<real>)`.

**Discussion** Returns the end angle of `elliptical-object`. If `elliptical-object` is a full ellipse or closed path then `ellipse-end-angle` returns `#f`; otherwise the value is a number greater than zero, and less than or equal to `2\pi`.

**See also**

• `make-ellipse`

**ellipse-radii** Generic function

Returns four values corresponding to the two radius vectors of an elliptical arc.

**Signature** `ellipse-radii elliptical-object => r1-dx r1-dy r2-dx d2-dy`

**Parameters**

• **elliptical-object** – An instance of type `type-union(<ellipse>, <elliptical-arc>)`.

**Values**

• **r1-dx** – An instance of type `<real>`.
• **r1-dy** – An instance of type `<real>`.
• **r2-dx** – An instance of type `<real>`.
• **d2-dy** – An instance of type `<real>`.

**Discussion** Returns four values corresponding to the two radius vectors of `elliptical-object`. 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**

• `make-ellipse`

**ellipse-start-angle** Generic function

Returns the start angle of an elliptical arc.

**Signature** `ellipse-start-angle elliptical-object => angle`

**Parameters**

• **elliptical-object** – An instance of type `type-union(<ellipse>, <elliptical-arc>)`.

**Values**

• **angle** – An instance of type `false-or(<real>)`.

**Discussion** Returns the start angle of `elliptical-object`. If `elliptical-object` is a full ellipse or closed path then `ellipse-start-angle` returns `#f`; otherwise the value will be a number greater than zero, and less than or equal to `2\pi`.

**See also**

• `make-ellipse`

<elliptical-arc> Abstract Instantiable Class

An **elliptical arc** is a path consisting of all or a portion of the outline of an ellipse.
Superclasses  <path>

Init-Keywords

- center-x – An instance of type <real>.
- center-y – An instance of <real>.
- center-point – An instance of type <point>.
- radius-1-dx – An instance of <real>.
- radius-1-dy – An instance of <real>.
- radius-2-dx – An instance of <real>.
- radius-2-dy – An instance of <real>.
- start-angle – An instance of false-or(<real>).
- end-angle – An instance of false-or(<real>).

Discussion  An elliptical arc is a path consisting of all or a portion of the outline of an ellipse. Circular arcs are special cases of elliptical arcs.

Operations

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

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

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

- box-edges
- transform-region

See also

- elliptical-arc?
- make-elliptical-arc

elliptical-arc?  Generic function

Returns #t if an object is an elliptical arc.

Signature  elliptical-arc? object => boolean

Parameters

- object – An instance of type <object>.

Values

- boolean – An instance of type <boolean>.

Discussion  Returns #t if object is an elliptical arc, otherwise returns #f.

See also
<line> Abstract Instantiable Class

The class that corresponds to a line.

**Superclasses** <path>

**Init-Keywords**

- **start-x** – An instance of <real>.
- **start-y** – An instance of <real>.
- **end-x** – An instance of <real>.
- **end-y** – An instance of <real>.
- **points** – Instances of <point>.

**Discussion**

The class that corresponds to a line. This is a subclass of <path>.

This is the instantiable class that implements a line segment. **make-line** instantiates an object of type <line>.

**Operations**

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

- 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?

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

- box-edges
- transform-region

**See also**

- <path>
- make-line

**line?** Generic function

Returns #t if an object is a line.

**Signature** line? object => boolean

**Parameters**
• **object** – An instance of type `<object>`.

**Values**

• **boolean** – An instance of type `<boolean>`.

**Discussion** Returns `#t` if `object` is a line, otherwise returns `#f`.

### line-end-point Generic function

Returns the ending point of a line.

**Signature** `line-end-point line => point`

**Parameters**

• **line** – An instance of type `<line>`.

**Values**

• **point** – An instance of type `<point>`.

**Discussion** Returns the ending point of `line` as a `<point>` object.

**See also**

• `line-start-point`

### line-end-position Generic function

Returns the ending point of a line.

**Signature** `line-end-position line => x y`

**Parameters**

• **line** – An instance of type `<line>`.

**Values**

• **x** – An instance of type `<real>`.

• **y** – An instance of type `<real>`.

**Discussion**

Returns two real numbers representing the `x` and `y` coordinates of the ending point of `line`.

The arguments `x` and `y` represent the `x` and `y` coordinates of the end of the line, respectively.

**See also**

• `line-start-position`

### line-start-point Generic function

Returns the starting point of a line.

**Signature** `line-start-point line => point`

**Parameters**

• **line** – An instance of type `<line>`.

**Values**

• **point** – An instance of type `<point>`.

**Discussion** Returns the starting point of `line` as a `<point>` object.

**See also**

• `line-end-point`
**line-start-position** Generic function

Returns the starting point of a line.

**Signature**

\[ \text{line-start-position } \text{line} \Rightarrow x \ y \]

**Parameters**

- \text{line} – An instance of type \text{<line>}.  

**Values**

- \text{x} – An instance of type \text{<real>}.  
- \text{y} – An instance of type \text{<real>}.  

**Discussion**

Returns two real numbers representing the \text{x} and \text{y} coordinates of the starting point of \text{line}.  

The arguments \text{x} and \text{y} represent the \text{x} and \text{y} coordinates of the start of the line, respectively.

See also

- \text{line-end-position}

**make-ellipse** Function

Returns an object of class \text{<ellipse>}.  

**Signature**

\[ \text{make-ellipse} \text{ center-x } \text{ center-y } \text{ radius-1-dx } \text{ radius-1-dy } \text{ radius-2-dx } \text{ radius-2-dy } \#key \text{ start-angle } \text{ end-angle} \Rightarrow \text{ellipse} \]

**Signature**

\[ \text{make-ellipse*} \text{ center-point } \text{ radius-1-dx } \text{ radius-1-dy } \text{ radius-2-dx } \text{ radius-2-dy } \#key \text{ start-angle } \text{ end-angle} \Rightarrow \text{ellipse} \]

**Parameters**

- \text{radius-1-dx} – An instance of type \text{<real>}.  
- \text{radius-1-dy} – An instance of type \text{<real>}.  
- \text{radius-2-dx} – An instance of type \text{<real>}.  
- \text{radius-2-dy} – An instance of type \text{<real>}.  
- \text{start-angle} – An instance of type \text{false-or(<real>)}.  
- \text{end-angle} – An instance of type \text{false-or(<real>)}.  

The following arguments are specific to \text{make-ellipse}.

**Parameters**

- \text{center-x} – An instance of type \text{<real>}.  
- \text{center-y} – An instance of type \text{<real>}.  

The following argument is specific to \text{make-ellipse}.

**Parameters**

- \text{center-point} – An instance of type \text{<point>}.  

**Values**

- \text{ellipse} – An instance of type \text{<ellipse>}.  

**Discussion**

Returns an object of class \text{<ellipse>}. The center of the ellipse is at the position \text{center-x}*\text{center-y}* or the point \text{center-point}.  

4.3. DUIM-Extended-Geometry Module
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 \textit{ellipse-not-well-defined} error is signalled. The special case of an ellipse with its axes aligned with the coordinate axes can be obtained by setting both \(radius-1-dy\) and \(radius-2-dx\) to 0.

If \textit{start-angle} or \textit{end-angle} are supplied, the ellipse is the \textit{pie slice} 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 \textit{start-angle} to \textit{end-angle}. Angles are measured counter-clockwise with respect to the positive \(x\) axis. If \textit{end-angle} is supplied, the default for \textit{start-angle} is 0; if \textit{start-angle} is supplied, the default for \textit{end-angle} is \(2\pi\); if neither is supplied then the region is a full ellipse and the angles are meaningless.

The function \textit{make-ellipse*} is identical to \textit{make-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

- \texttt{<ellipse>}

\textbf{make-elliptical-arc Function}

Returns an object of class \texttt{<elliptical-arc>}. 

\textbf{Signature} \texttt{make-elliptical-arc center-x center-y radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key start-angle end-angle => arc}

\textbf{Signature} \texttt{make-elliptical-arc* center-point radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key start-angle end-angle => arc}

\textbf{Parameters}

- \texttt{radius-1-dx} – An instance of type \texttt{<real>}. 
- \texttt{radius-1-dy} – An instance of type \texttt{<real>}. 
- \texttt{radius-2-dx} – An instance of type \texttt{<real>}. 
- \texttt{radius-2-dy} – An instance of type \texttt{<real>}. 
- \texttt{start-angle} – An instance of type \texttt{false-or(<real>)}. 
- \texttt{end-angle} – An instance of type \texttt{false-or(<real>)}. 

The following arguments are specific to \textit{make-elliptical-arc}.

\textbf{Parameters}

- \texttt{center-x} – An instance of type \texttt{<real>}. 
- \texttt{center-y} – An instance of type \texttt{<real>}. 

The following argument is specific to \textit{make-elliptical-arc*}.

\textbf{Parameters}

- \texttt{center-point} – An instance of type \texttt{<point>}. 

\textbf{Values}

- \texttt{arc} – An instance of type \texttt{<elliptical-arc>}. 

\textbf{Discussion}

Returns an object of class \texttt{<elliptical-arc>}. The center of the ellipse is at the position \texttt{center-x,center-y} or the point \texttt{center-point}.
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 collinear, the ellipse is not well-defined and the \textit{ellipse-not-well-defined} 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 \textit{radius-1-dy} and \textit{radius-2-dx} to 0.

If \textit{start-angle} and \textit{end-angle} are supplied, the arc is swept from \textit{start-angle} to \textit{end-angle}. Angles are measured counter-clockwise with respect to the positive \textit{x} axis. If \textit{end-angle} is supplied, the default for \textit{start-angle} is 0; if \textit{start-angle} is supplied, the default for \textit{end-angle} is \(2\pi\); if neither is supplied then the region is a closed elliptical path and the angles are meaningless.

The function \textit{make-elliptical-arc*} is identical to \textit{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**

- \texttt{<elliptical-arc>}

**make-line Function**

Returns an object of class \texttt{<line>}.t

Signature

\[
\begin{align*}
\text{make-line} & \quad \text{start-x start-y end-x end-y} \Rightarrow \text{line} \\
\text{make-line*} & \quad \text{start-point end-point} \Rightarrow \text{line}
\end{align*}
\]

Parameters

- \texttt{start-x} – An instance of type \texttt{<real>}. \\
- \texttt{start-y} – An instance of type \texttt{<real>}. \\
- \texttt{end-x} – An instance of type \texttt{<real>}. \\
- \texttt{end-y} – An instance of type \texttt{<real>}. \\
- \texttt{start-point} – An instance of type \texttt{<point>}. \\
- \texttt{end-point} – An instance of type \texttt{<point>}. \\

Values

- \texttt{line} – An instance of type \texttt{<line>}. \\

Discussion

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

**make-polygon Function**

Returns an object of class \texttt{<polygon>}.t

Signature

\[
\begin{align*}
\text{make-polygon} & \quad \text{coord-seq} \Rightarrow \text{polygon} \\
\text{make-polygon*} & \quad \text{point-seq} \Rightarrow \text{polygon}
\end{align*}
\]

The following argument is specific to \textit{make-polygon}.

Parameters

- \texttt{coord-seq} – An instance of type \texttt{limited(<sequence>, of: <real>).} \\

The following argument is specific to \textit{make-polygon*}.

Parameters

- \texttt{point-seq} – An instance of type \texttt{limited(<sequence>, of: <point>).} \\

Values
• polygon – An instance of type <polygon>.

Discussion

Returns an object of class <polygon> consisting of the area contained in the boundary that is specified by 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.

The function make-polygon* is identical to make-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.

make-polyline Function

Returns an object of class <polyline>.

Signature make-polyline coord-seq #key closed? => polyline

Signature make-polyline* point-seq #key closed? => polyline

Parameters

• closed? – An instance of type <boolean>. Default value: #f.

The following argument is specific to make-polyline.

Parameters

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

The following argument is specific to make-polyline*.

Parameters

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

Values

• polyline – An instance of type <polyline>

Discussion

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.

make-rectangle Function

Returns an object of class <rectangle>.

Signature make-rectangle x1 y1 x2 y2 => rectangle

Signature make-rectangle* min-point max-point => rectangle

The following arguments are specific to make-rectangle.

Parameters

• x1 – An instance of type <real>. The x coordinate of the left top of the rectangle.
• $y_1$ – An instance of type `<real>`. The $y$ coordinate of the left top of the rectangle.
• $x_2$ – An instance of type `<real>`. The $x$ coordinate of the bottom right of the rectangle.
• $y_2$ – An instance of type `<real>`. The $y$ coordinate of the bottom right of the rectangle.

The following arguments are specific to `make-rectangle\*`.

**Parameters**

• `min-point` – The minimum point (left top) of the rectangle.
• `max-point` – The maximum point (bottom right) of the rectangle.

**Values**

• `rectangle` – An instance of type `<rectangle>`.

**Discussion**

Returns an object of class `<rectangle>` whose edges are parallel to the coordinate axes. One corner is at the point $point1$ or the position $x_1,y_1$ and the opposite corner is at the point $point2$ or the position $x_2,y_2$. There are no ordering constraints among $point1$ and $point2$ or $x_1$ and $x_2$, and $y_1$ and $y_2$.

The function `make-rectangle\*` is identical to `make-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.

**<polygon> Abstract Instantiable Class**

The class that corresponds to a polygon.

**Superclasses** `<area>`

**Init-Keywords**

• `coordinates` – An instance of type `limited(<sequence>, of: <real>)`.
• `points` – An instance of type `limited(<sequence>, of: <real>)`.

**Discussion**

The class that corresponds to a polygon. This is a subclass of `<area>`.

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 `coordinates`: init-keyword, which is a sequence of real numbers, with $x$ and $y$ coordinates alternating within the sequence.

To describe a polygon in terms of composite point objects, use the `points`: init-keyword, which is a sequence of instances of `<point>`. You should be aware that using composite points may lead to a loss of performance.

Exactly one of `coordinates`: and `points`: is required.

**Operations**

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

• `do-polygon-coordinates`
• `do-polygon-segments`
• `draw-design`
• `polygon?`
• `polygon-coordinates`
• polygon-points

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

• box-edges
• transform-region

See also

• <area>
• make-polygon
• polygon?
• polygon-coordinates
• polygon-points

polygon? Generic function
Returns #t if its argument is a polygon.

Signature  polygon? object => boolean

Parameters

• object – An instance of type <object>.

Values

• boolean – An instance of type <boolean>.

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

See also

• <polygon>
• polygon-coordinates
• polygon-points

polygon-coordinates Generic function
Returns a sequence of coordinate pairs that specify the segments in a polygon or a polyline.

Signature  polygon-coordinates polygon-or-polyline => coordinates

Parameters

• polygon-or-polyline – An instance of type type-union(<polygon>, <polyline>).

Values

• coordinates – An instance of type limited(<sequence>, of: <real>).

Discussion Returns a sequence of coordinate pairs that specify the segments in polygon-or-polyline.

See also

• <polygon>
• polygon?
• polygon-points

polygon-points Generic function
Returns a sequence of points that specify the segments in a polygon or a polyline.
Signature  polygon-points $\text{polygon-or-polyline} \Rightarrow \text{points}$

Parameters

- **$\text{polygon-or-polyline}$** – An instance of type $\text{type-union}(<\text{polygon}, <\text{polyline}>)$.

Values

- **$\text{points}$** – An instance of type $\text{limited}(<\text{sequence}, \text{of}: <\text{point}>)$

Discussion  Returns a sequence of points that specify the segments in $\text{polygon-or-polyline}$.

See also

- <$\text{polygon}$>
- $\text{polygon?}$
- $\text{polygon-coordinates}$

$\text{<polyline>}$ Abstract Instantiable Class

The protocol class that corresponds to a polyline.

Superclasses  $\text{<path>}$

Init-Keywords

- **$\text{coordinates}$** – An instance of type $\text{limited}(<\text{sequence}, \text{of}: <\text{real}>)$ Required.
- **$\text{points}$** – An instance of type $\text{limited}(<\text{sequence}, \text{of}: <\text{real}>)$ Required.

Discussion  The protocol class that corresponds to a polyline.

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 $\text{coordinates}$: init-keyword, which is a sequence of real numbers, with x and y coordinates alternating within the sequence.

To describe a polyline in terms of composite point objects, use the $\text{points}$: init-keyword, which is a sequence of instances of $\text{<point>}$. You should be aware that using composite points may lead to a loss of performance.

Exactly one of $\text{coordinates}$: and $\text{points}$: is required.

Operations  The following operations are exported from the $\text{DUIM-Extended-Geometry}$ module.

- $\text{do-polygon-coordinates}$
- $\text{do-polygon-segments}$
- $\text{draw-design}$
- $\text{polygon-coordinates}$
- $\text{polygon-points}$
- $\text{polyline}$
- $\text{polyline-closed}$?

The following operations are exported from the $\text{DUIM-Geometry}$ module.

- $\text{box-edges}$
• transform-region

See also
• <path>
• <point>
• make-polyline
• polyline?
• polyline-closed?

polyline? Generic function
Returns #t if an object is a polyline.

Signature polyline? object => boolean

Parameters
• object – An instance of type <object>.

Values
• boolean – An instance of type <boolean>.

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

See also
• <polyline>
• polyline-closed?

polyline-closed? Generic function
Returns #t if the polyline is closed.

Signature polyline-closed? polyline => boolean

Parameters
• polyline – An instance of type <polyline>.

Values
• boolean – An instance of type <boolean>.

Discussion Returns #t if the polyline polyline is closed, otherwise returns #f. This function need be implemented only for polylines, not for polygons.

See also
• <polyline>
• polyline?

<rectangle> Abstract Instantiable Class
The protocol class that corresponds to a rectangle.

Superclasses <area>

Init-Keywords
• min-x – An instance of type <real>.
• min-y – An instance of type <real>.
• max-x – An instance of type <real>. 
• **max-y** – An instance of type `<real>`.
• **points** – An instance of type `limited(<sequence>, of: <point>)`.

**Discussion**

The protocol class that corresponds to a rectangle. This is a subclass of `<polygon>`. Rectangles whose edges are parallel to the coordinate axes are a special case of polygon that can be specified completely by four real numbers \(x_1,y_1,x_2,y_2\). They are not closed under general affine transformations (although they are closed under rectilinear transformations).

**Operations**

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

• `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`

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

• `box-edges`
• `transform-region`

**See also**

• `<polygon>`
• `make-rectangle`
• `rectangle?`
• `rectangle-edges`
• `rectangle-height`
• `rectangle-max-point`
• `rectangle-max-position`
• `rectangle-min-point`
• `rectangle-min-position`
• `rectangle-size`
• rectangle-width

rectangle? Generic function
Returns #t if the object is a rectangle.

Signature rectangle? object => boolean

Parameters
• object – An instance of type <object>.

Values
• boolean – An instance of type <boolean>.

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

See also
• <rectangle>
• rectangle-edges
• rectangle-height
• rectangle-max-point
• rectangle-max-position
• rectangle-min-point
• rectangle-min-position
• rectangle-size
• rectangle-width

rectangle-edges Generic function
Returns the coordinates of the minimum and maximum of the rectangle.

Signature rectangle-edges rectangle => x1 y1 x2 y2

Parameters
• rectangle – An instance of type <rectangle>.

Values
• min-x – An instance of type <real>.
• min-y – An instance of type <real>.
• max-x – An instance of type <real>.
• max-y – An instance of type <real>.

Discussion
Returns the coordinates of the minimum x and y and maximum x and y of the rectangle rectangle as four values, min-x, min-y, max-x, and max-y.

The argument min-x represents the x coordinate of the top left of the rectangle.

The argument min-y represents the y coordinate of the top left of the rectangle.

The argument max-x represents the x coordinate of the bottom right of the rectangle.

The argument max-y represents the y coordinate of the bottom right of the rectangle.

See also
rectangle-height Generic function
Returns height of the rectangle.

Signature  rectangle-height rectangle => height

Parameters
• rectangle – An instance of type <rectangle>.

Values
• height – An instance of type <real>.

Discussion  Returns the height of the rectangle, which is the difference between the maximum y and its minimum y.

See also
• <rectangle>
• rectangle?
• rectangle-height
• rectangle-max-point
• rectangle-max-position
• rectangle-min-point
• rectangle-min-position
• rectangle-size
• rectangle-width

dictionary

rectangle-max-point Generic function
Returns the bottom right point of the rectangle.

Signature  rectangle-max-point rectangle => point

Parameters
• rectangle – An instance of type <rectangle>.

Values
• point – An instance of type <point>.

Discussion  Returns the bottom right point of the rectangle.

See also
• `<rectangle>`
• `rectangle?`
• `rectangle-edges`
• `rectangle-height`
• `rectangle-max-position`
• `rectangle-min-point`
• `rectangle-min-position`
• `rectangle-size`
• `rectangle-width`

`rectangle-max-position` Generic function
Returns the x and y coordinates of the bottom right of the rectangle.

**Signature** `rectangle-max-position rectangle => x2 y2`

**Parameters**
• `rectangle` – An instance of type `<rectangle>`.

**Values**
• `x2` – An instance of type `<real>`.
• `y2` – An instance of type `<real>`.

**Discussion** Returns the x and y coordinates of the bottom right of the rectangle.

**See also**
• `<rectangle>`
• `rectangle?`
• `rectangle-edges`
• `rectangle-height`
• `rectangle-max-point`
• `rectangle-min-point`
• `rectangle-min-position`
• `rectangle-size`
• `rectangle-width`

`rectangle-min-point` Generic function
Returns the left top point of the rectangle.

**Signature** `rectangle-min-point rectangle => point`

**Parameters**
• `rectangle` – An instance of type `<rectangle>`.

**Values**
• `point` – An instance of type `<point>`.

**Discussion** Returns the left top point of the rectangle.

**See also**
• <rectangle>
• rectangle?
• rectangle-edges
• rectangle-height
• rectangle-max-point
• rectangle-max-position
• rectangle-min-position
• rectangle-min-point
• rectangle-size
• rectangle-width

rectangle-min-position Generic function
Returns the x and y coordinates of the left top of the rectangle.

Signature rectangle-min-position rectangle => x1 y1

Parameters
• rectangle – An instance of type <rectangle>.

Values
• x1 – An instance of type <real>.
• y1 – An instance of type <real>.

Discussion Returns the x and y coordinates of the left top of the rectangle.

See also
• <rectangle>
• rectangle?
• rectangle-edges
• rectangle-height
• rectangle-max-point
• rectangle-max-position
• rectangle-min-point
• rectangle-size
• rectangle-width

rectangle-size Generic function
Returns the width and the height of the rectangle.

Signature rectangle-size rectangle => width height

Parameters
• rectangle – An instance of type <rectangle>.

Values
• width – An instance of type <real>.
• height – An instance of type <real>.

Discussion Returns two values, the width and the height.
See also

- `<rectangle>`
- `rectangle?`
- `rectangle-edges`
- `rectangle-height`
- `rectangle-max-point`
- `rectangle-max-position`
- `rectangle-min-point`
- `rectangle-min-position`
- `rectangle-width`

`rectangle-width` Generic function

Returns the width of the rectangle.

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

**Parameters**

- `rectangle` – An instance of type `<rectangle>`.

**Values**

- `width` – An instance of type `<real>`.

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

See also

- `<rectangle>`
- `rectangle?`
- `rectangle-edges`
- `rectangle-height`
- `rectangle-max-point`
- `rectangle-max-position`
- `rectangle-min-point`
- `rectangle-min-position`
- `rectangle-size`
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. *DUIM-DCs Module* 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.

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 the following table

<table>
<thead>
<tr>
<th>&lt;object&gt;</th>
<th>&lt;pen&gt;</th>
<th>&lt;brush&gt;</th>
<th>&lt;palette&gt;</th>
<th>&lt;ink&gt;</th>
<th>&lt;text-style&gt;</th>
<th>&lt;device-font&gt;</th>
</tr>
</thead>
</table>

- *<pen>* 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.
- *<brush>* The protocol class for brushes. Brushes are used to fill in 2 dimensional areas with a specific color or pattern.
- *<palette>* The protocol class for palettes. A palette provides a set of colors which can be made available to an application.
- *<ink>* 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 Subclasses of *<ink>*.
- *<text-style>* 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>`.

**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>`.

**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.

**DUIM-DCs Module**

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

= Generic function

Returns `#t` if two objects are equal.

**Signature** = `color1 color2 => boolean`

**Signature** = `pen1 pen2 => boolean`

**Signature** = `brush1 brush2 => boolean`

**Signature** = `text-style1 text-style2 => value`

**Parameters**

• `color1` – An instance of type `<color>`.

• `color2` – An instance of type `<color>`.

• `pen1` – An instance of type `<pen>`.

• `pen2` – An instance of type `<pen>`.

• `brush1` – An instance of type `<brush>`.

• `brush2` – An instance of type `<brush>`.

• `text-style1` – An instance of type `<text-style>`.
• **text-style2** – An instance of type `<text-style>`.

**Values**

• **boolean** – An instance of type `<boolean>`.

**Discussion** Returns `#t` if two objects are equal.

### add-colors

**Generic function**

Adds one or more colors to a palette and returns the updated palette.

**Signature**

`add-colors palette #rest colors => palette`

**Parameters**

• **palette** – An instance of type `<palette>`.

• **colors** – Instances of type `<color>`.

**Values**

• **palette** – An instance of type `<palette>`.

**Discussion** Adds `colors` to `palette` and returns the updated palette.

### $background

**Constant**

An indirect ink that uses the medium’s background design.

**Type** `<ink>`

**Discussion** An indirect ink that uses the medium’s background design.

**See also**

• `<palette>`

### $black

**Constant**

The usual definition of black.

**Type** `<color>`

**Discussion** The usual definition black, the absence of all colors. In the `rgb` color model, its value is `000`.

**See also**

• `<color>`

### $blue

**Constant**

The usual definition of the color blue.

**Type** `<color>`

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

**See also**

• `<color>`

### $boole-clr

**Constant**

The logical operator that is always 0.

**Type** `<integer>`

**Discussion** The logical operator that is always 0. It is a suitable first argument to the `boole` function.
$boole-set Constant
The logical operator that is always 1.

Type <integer>

Discussion The logical operator that is always 1. It is a suitable first argument to the boole function.

$boole-1 Constant
The logical operator that is always the same as the first integer argument to the boole function.

Type <integer>

Discussion The logical operator that is always the same as the first integer argument to the boole function. It is a suitable first argument to the boole function.

$boole-2 Constant
The logical operator that is always the same as the second integer argument to the boole function.

Type <integer>

Discussion The logical operator that is always the same as the second integer argument to the boole function. It is a suitable first argument to the boole function.

$boole-c1 Constant
The logical operator that is always the same as the complement of the first integer argument to the boole function.

Type <integer>

Discussion The logical operator that is always the same as the complement of the first integer argument to the boole function. It is a suitable first argument to the boole function.

$boole-c2 Constant
The logical operator that is always the same as the complement of the second integer argument to the boole function.

Type <integer>

Discussion The logical operator that is always the same as the complement of the second integer argument to the boole function. It is a suitable first argument to the boole function.

$boole-and Constant
The logical operator and.

Type <integer>

Discussion The logical operator and. It is a suitable first argument to the boole function.

$boole-ior Constant
The logical operator inclusive or.

Type <integer>

Discussion The logical operator inclusive or. It is a suitable first argument to the boole function.

$boole-xor Constant
The logical operator exclusive or.

Type <integer>

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

$boole-eqv Constant
The logical operator equivalence (exclusive nor).

Type <integer>
Discussion  The logical operator equivalence (exclusive nor). It is a suitable first argument to the boole function.

$\texttt{boole-nand}$ Constant

The logical operator not-and.

Type  <integer>

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

$\texttt{boole-nor}$ Constant

The logical operator not-or.

Type  <integer>

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

$\texttt{boole-andc1}$ Constant

The logical operator that is the and of the complement of the first integer argument to the boole function with the second.

Type  <integer>

Discussion  The logical operator that is the and of the complement of the first integer argument to the boole function with the second. It is a suitable first argument to the boole function.

$\texttt{boole-andc2}$ Constant

The logical operator that is the and of the first integer argument to the boole function with the second with the complement of the second.

Type  <integer>

Discussion  The logical operator that is and of the first integer argument to the boole function with the complement of the second. It is a suitable first argument to the boole function.

$\texttt{boole-orc1}$ Constant

The logical operator that is the or of the complement of the first integer argument to the boole function with the second.

Type  <integer>

Discussion  The logical operator that is the or of the complement of the first integer argument to the boole function with the second. It is a suitable first argument to the boole function.

$\texttt{boole-orc2}$ Constant

The logical operator that is the or of the first integer argument to the boole function with the second with the complement of the second.

Type  <integer>

Discussion  The logical operator that is or of the first integer argument to the boole function with the complement of the second. It is a suitable first argument to the boole function.

$\texttt{bricks-stipple}$ Constant

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  <array>

Discussion  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

- brush-stipple

5.3. DUIM-DCs Module
Abstract Instantiable Class

The protocol class for brushes.

Superclasses <object>

Init-Keywords

• foreground – An instance of type <ink>.
• background – An instance of type <ink>.
• mode – An instance of type <integer>.
• fill-style – An instance of type false-or(<integer>). Default value: #f.
• fill-rule – An instance of type false-or(<integer>). Default value: #f.
• tile – An instance of type false-or(<integer>). Default value: #f.
• stipple – An instance of type false-or(<integer>). Default value: #f.
• ts-x – An instance of false-or(<integer>). Default value: #f.
• ts-y – An instance of false-or(<integer>). Default value: #f.

Discussion The protocol class for brushes.

Operations

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

• =
• 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

See also

• make

brush? Generic function

Returns #t if its argument is a brush.

Signature brush? object => boolean

Parameters

• object – An instance of type <object>.

Values

• boolean – An instance of type <boolean>.
Discussion  Returns #t if its argument is a brush.

**brush-background** Generic function

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

**Signature**  brush-background brush => ink

**Parameters**

- brush – An instance of type <brush>.

**Values**

- ink – An instance of type <ink>.

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

See also

- brush-fill-rule

**brush-fill-rule** Generic function

Returns the fill rule of the brush.

**Signature**  brush-fill-rule brush => fill-rule

**Parameters**

- brush – An instance of type <brush>.

**Values**

- fill-rule – An instance of type fill-rule or <boolean>.

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

See also

- brush-fill-style

**brush-fill-style** Generic function

Returns the fill style of the brush.

**Signature**  brush-fill-style brush => fill-style

**Parameters**

- brush – An instance of type <brush>.

**Values**

- fill-style – An instance of type fill-style or <boolean>.

**Discussion**  Returns the fill style of brush, or #f, if brush does not have a fill style.

See also

- brush-fill-rule

**brush-foreground** Generic function

Returns the ink that is the foreground color of a brush.

**Signature**  brush-foreground brush => ink

**Parameters**

- brush – An instance of type <brush>.

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

Discussion Returns the ink that is the foreground color of brush.
See also
  • brush-stipple

brush-mode Generic function
Returns an integer representing the drawing mode of a brush.

Signature brush-mode brush => integer
Parameters
  • brush – An instance of type <brush>.
Values
  • integer – An instance of type <integer>. Default value: $boole-1.
Discussion Returns an integer representing the drawing mode of brush.
See also
  • $boole-1

brush-stipple Generic function
Returns the stipple pattern of a brush.

Signature brush-stipple brush => stipple
Parameters
  • brush – An instance of type <brush>.
Values
  • stipple – A (stipple) or #f.
Discussion Returns the stipple pattern of brush.
See also
  • brush-tile
  • brush-fill-rule
  • brush-fill-style

brush-stretch-mode Generic function
Returns the stretch mode of the brush.

Signature brush-stretch-mode brush => stretch-mode
Parameters
  • brush – An instance of type <brush>.
Values
  • stretch-mode – An instance of stretch-mode or <boolean>.
Discussion Returns the stretch mode of the brush.

brush-tile Generic function
Returns the tile pattern of a brush.

Signature brush-tile brush => image
Parameters
  • **brush** – An instance of type `<brush>`.

Values
  • **image** – An instance of type `<image>`.

Discussion Returns the tile pattern of `brush`.

See also
  • `brush-stipple`
  • `brush-ts-x`
  • `brush-ts-y`

**brush-ts-x Generic function**
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`

**Parameters**
  • **brush** – An instance of type `<brush>`.

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

Discussion 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`

**brush-ts-y Generic function**
Returns the value of the y coordinate that is used to align the brush’s tile or stipple pattern.

**Signature** `brush-ts-y brush => value`

**Parameters**
  • **brush** – An instance of type `<brush>`.

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

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

See also
  • `brush-ts-x`

**<color> Abstract Instantiable Class**
The protocol class for colors.

**Superclasses** `<ink>`

**Init-Keywords**
  • **red** – An instance of type `<real>`.
  • **green** – An instance of type `<real>`.
  • **blue** – An instance of type `<real>`.
• **intensity** – An instance of type `limited(real, min: 0, max: \(sqrt(3)\))`.

• **hue** – An instance of type `limited(real, min: 0, max: 1)`.

• **saturation** – An instance of type `limited(real, min: 0, max: 1)`.

• **opacity** – An instance of type `limited(real, min: 0, max: 1)`.

**Discussion**

The `<color>` class is the protocol class for a color, and is a subclass of `<ink>`. A member of the class `<color>` 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.

The red:, green:, and blue: 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.

The intensity describes the brightness of the color. An intensity of 0 is black.

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.

The saturation describes the amount of white in the color. This is what distinguishes pink from red.

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.

All of the standard instantiable color classes provided by DUIM are immutable.

A color can be specified by four real numbers between 0 and 1 (inclusive), giving the amounts of red, green, blue, and opacity (alpha). 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.

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?`
- `color-ihs`
- `color-luminosity`
color? Generic function
Returns #t if object is a color.

Signature  
\texttt{color? object} \Rightarrow \texttt{boolean}

Parameters

- \texttt{object} – An instance of type \texttt{<object>}.

Values

- \texttt{boolean} – An instance of type \texttt{<boolean>}.

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

See also

- \texttt{<color>}
- \texttt{color-ihs}
- \texttt{color-luminosity}
- \texttt{<color-not-found>}
- \texttt{color-palette?}
- \texttt{color-rgb}

color-ihs Generic function
Returns four values, the intensity, hue, saturation, and opacity components of a color.

Signature  
\texttt{color-ihs color} \Rightarrow \texttt{intensity hue saturation opacity}

Parameters

- \texttt{color} – An instance of type \texttt{<color>}.

Values

- \texttt{intensity} – An instance of type \texttt{limited(<real>, min: 0, max: \sqrt{3})}.
- \texttt{hue} – An instance of type \texttt{limited(<real>, min: 0, max: 1)}.
- \texttt{saturation} – An instance of type \texttt{limited(<real>, min: 0, max: 1)}.
- \texttt{opacity} – An instance of type \texttt{limited(<real>, min: 0, max: 1)}.

Discussion  Returns four values, the \textit{intensity}, \textit{hue}, \textit{saturation}, and \textit{opacity} components of the color \texttt{color}. The first value is a real number between 0 and \sqrt{3} (inclusive). The second and third values are real numbers between 0 and 1 (inclusive).

See also

- \texttt{<color>}
- \texttt{color?}
- \texttt{color-luminosity}
- \texttt{color-palette?}
- `color-rdb`

**color-luminosity** **Generic function**
Returns the brightness of a color.

*Signature*  
`color-luminosity* color => luminosity`

*Parameters*
- `color` – An instance of type `<color>`.

*Values*
- `luminosity` – An instance of type `limited(<real>, min: 0, max: 1)`.

*Discussion*  
Returns the brightness of color `color` 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*
- `<color>`
- `color?`
- `color-ihs`
- `color-palette?`
- `color-rdb`

**<color-not-found>** **Concrete Sealed Class**
The class of the error that is signalled when a color that is not available is requested.

*Superclasses*  
`<error>`

*Init-Keywords*
- `color` – An instance of type `<color>`.

*Discussion*  
The class of the error that is signalled when a color that is not available is requested. The `color: init-keyword` is used to specify the color that was requested but was not available.

*Operations*
- None.

*See also*
- `<color>`
- `find-color`
- `remove-colors`
- `find-color`

**color-palette?** **Generic function**
Returns `#t` if the stream or medium supports color.

*Signature*  
`color-palette? palette => boolean`

*Parameters*
- `palette` – An instance of type `<palette>`.

*Values*
- `boolean` – An instance of type `<boolean>`.

*Discussion*  
Returns `#t` if the stream or medium supports color.
See also

- `<color>`
- `color?`
- `color-ihs`
- `color-luminosity`
- `color-rgb`

color-rgb Generic function

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

Signature  color-rgb color => ref green blue opacity

Parameters

- **color** – An instance of type `<color>`.

Values

- **red** – An instance of type `limited(<real>, min: 0, max: 1)`
- **green** – An instance of type `limited(<real>, min: 0, max: 1)`
- **blue** – An instance of type `limited(<real>, min: 0, max: 1)`
- **opacity** – An instance of type `limited(<real>, min: 0, max: 1)`.

Discussion  Returns four values, the red, green, blue, and opacity components of the color `color`. The values are real numbers between 0 and 1 (inclusive).

See also

- `<color>`
- `color?`
- `color-ihs`
- `color-luminosity`
- `color-rgb`

color-rgb Generic function

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

Signature  color-rgb color => ref green blue opacity

Parameters

- **color** – An instance of type `<color>`.

Values

- **red** – An instance of type `limited(<real>, min: 0, max: 1)`
- **green** – An instance of type `limited(<real>, min: 0, max: 1)`
- **blue** – An instance of type `limited(<real>, min: 0, max: 1)`
- **opacity** – An instance of type `limited(<real>, min: 0, max: 1)`.

Discussion  Returns four values, the red, green, blue, and opacity components of the color `color`. The values are real numbers between 0 and 1 (inclusive).

See also

- `<color>`
- `color?`
- `color-ihs`
- `color-luminosity`
- `color-rgb`

contrastings-colors-limit Generic function

Returns the number of contrasting colors that can be rendered on the current platform.

Signature  contrasting-colors-limit port => integer

Parameters

- **port** – An instance of type `<silica>`

Values

- **integer** – An instance of type `<integer>`.

Discussion  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 `port`. 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

- `contrasting-dash-patterns-limit`
- `make-contrasting-colors`
**contrasting-dash-patterns-limit** Generic function
Returns the number of contrasting dash patterns that the specified port can generate.

**Signature**
contrasting-dash-patterns-limit port => no-of-patterns

**Parameters**
- port – An instance of type <silica>.

**Values**
- no-of-patterns – An instance of type <integer>.

**Discussion**
Returns the number of contrasting dash patterns that the specified port can generate.

**See also**
- contrasting-colors-limit
- make-contrasting-dash-patterns

$cross-hatch Constant
A stipple pattern for use in creating a patterned brush with alternating solid and dashed lines.

**Type** <array>

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

**See also**
- <color>.

$cyan Constant
The usual definition for the color cyan.

**Type** <color>

**Discussion**
The usual definition for the color cyan.

**See also**
- <color>.

$dash-dot-dot-pen Constant
A pen that draws a line with two dots between each dash.

**Type** <pen>

**Discussion**
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, 1].

**See also**
- <pen>
- $solid-pen
- $magenta
- $dash-dot-pen
- $dotted-pen

$dash-dot-pen Constant
A pen that draws a dashed and dotted line.

**Type** <pen>
Discussion A pen that draws a dashed and dotted line. The line width is 1 and dashes: is #[4, 1, 1, 1].

See also

- <pen>
- $solid-pen
- $magenta
- $dash-dot-pen
- $dotted-pen

$dashed-pen Constant
A pen that draws a dashed line.

Type <pen>

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

See also

- <pen>
- $solid-pen
- $magenta
- $dash-dot-pen
- $dotted-pen

default-background Generic function
Returns the ink that is the default background of its argument.

Signature default-foreground object => background

Parameters

- object – An instance of type <object>.

Values

- background – An instance of type <ink>.

Discussion Returns the ink that is the default background of its argument.

See also

- brush-fill-style
- default-background-setter
- default-foreground

default-background-setter Generic function
Sets the default background.

Signature default-foreground-setter background object => background

Parameters

- background – An instance of type <ink>.
- object – An instance of type <object>.

Values
• background – An instance of type *<ink>*.

**Discussion** Sets the default background for *object*.

**See also**

• *brush-fill-style*
  
  • *default-background*
  
  • *default-foreground-setter*

**default-foreground** Generic function

Returns the ink that is the default foreground of its argument.

**Signature** default-foreground *object => foreground*

**Parameters**

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

**Values**

• *foreground* – An instance of type *<ink>*.

**Discussion** Returns the ink that is the default foreground of its argument.

**See also**

• *brush-fill-rule*
  
  • *default-background*
  
  • *default-foreground-setter*

**default-foreground-setter** Generic function

Sets the default foreground.

**Signature** default-foreground-setter *foreground object => foreground*

**Parameters**

• *foreground* – An instance of type *<ink>*.
  
• *object* – An instance of type *<object>*.

**Values**

• *foreground* – An instance of type *<ink>*.

**Discussion** Sets the default foreground for *object*.

**See also**

• *brush-fill-rule*
  
  • *default-background-setter*
  
  • *default-foreground*

**default-text-style** Generic function

Returns the default text style for its argument.

**Signature** default-text-style *object => text-style*

**Parameters**

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

**Values**
- **text-style** – An instance of type `<text-style>`.

**Discussion** 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**
- `default-text-style-setter`

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

Sets the default text style.

**Signature** `default-text-style-setter text-style object => text-style`

**Parameters**
- `text-style` – An instance of type `<text-style>`.
- `object` – An instance of type `<object>`.

**Values**
- `text-style` – An instance of type `<text-style>`.

**Discussion** Sets the default text style.

**See also**
- `default-text-style`

**<device-font>** Concrete Sealed Class

The protocol class for device-specific fonts.

**Superclasses** `<text-style>`

**Init-Keywords**
- `port`
- `font-name`

**Discussion** The protocol class for device-specific fonts.

**Operations**
- None.

**See also**
- `<text-style>`

**$diagonal-hatch-down** Constant

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

**Type** `<array>`

**Discussion** 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`

**$diagonal-hatch-up** Constant

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

**Type** `<array>`
Discussion  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

$dotted-pen Constant
A pen that draws a dotted line.

Type  <pen>

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

See also  
  • <pen>
  • $solid-pen
  • $dash-dot-pen

find-color Generic function
Looks up and returns a color by name.

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

Parameters  
  • 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>.

Discussion  
Looks up and returns a color by name. This is a list of 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
• 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

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
• stencil?
• contrasting-dash-patterns-limit
• $black
• $red
• $yellow
• $green
• $blue
• $magenta

$foreground Constant
An indirect ink that uses the medium’s foreground design.

Type <ink>

Discussion An indirect ink that uses the medium’s foreground design.

See also
• <ink>
• <palette>

fully-merged-text-style? Generic function
Returns #:t if the specified text style is completely specified.

Signature fully-merged-text-style? text-style => boolean

Parameters
• text-style – An instance of type <text-style>.

Values
• boolean – An instance of type <boolean>.

Discussion Returns #:t if the specified text style is completely specified.

See also
• merge-text-styles
$\text{green Constant}$
The usual definition of the color green.

**Type** <color>

**Discussion** The usual definition of the color green.

**See also**

• <color>

$\text{hearts-stipple Constant}$
A stipple pattern for use in creating a patterned brush that draws a heart shape.

**Type** <array>

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

**See also**

• brush-stipple

$\text{horizontal-hatch Constant}$
A stipple pattern for use in creating a patterned brush with alternating horizontal rows of lines and spaces.

**Type** <array>

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

**See also**

• brush-stipple

<image> Abstract Class
The class for objects that are images.

**Superclasses** <ink>

**Discussion** The class for objects that are images.

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

• image?

The following operation is exported from the DUIM-Graphics module.

• <graphics>

**See also**

• image?
• image-depth
• image-height
• image-width
• <ink>

image? Generic function
Returns #t if its argument is an image.

**Signature** image? object => boolean

Parameters
• **object** – An instance of type `<object>`.

**Values**

• **boolean** – An instance of type `<boolean>`.

**Discussion** Returns `#t` if its argument is an image.

**See also**

• `<image>`
  • `image-depth`
  • `image-height`
  • `image-width`

**image-depth** Generic function

Returns the depth of an image.

**Signature** `image-depth image => depth`

**Parameters**

• **image** – An instance of type `<image>`.

**Values**

• **depth** – An instance of type `<real>`.

**Discussion** Returns the depth of the image `image`.

**See also**

• `<image>`
  • `image?`
  • `image-height`
  • `image-width`

**image-height** Generic function

Returns the height of an image.

**Signature** `image-height image => height`

**Parameters**

• **image** – An instance of type `<image>`.

**Values**

• **height** – An instance of type `<real>`.

**Discussion** Returns the height of the image `image`.

**See also**

• `<image>`
  • `image?`
  • `image-depth`
  • `image-width`

**image-width** Generic function

Returns the width of an image.
Signature  image-width image => width

Parameters

• image – An instance of type <image>.

Values

• width – An instance of type <real>.

Discussion  Returns the width of the image image.

See also

• <image>
  • image?
  • image-depth
  • image-height

<ink> Abstract Class

The class of objects that represent a way of arranging colors and opacities in the drawing plane.

Superclasses  <object>

Discussion  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 DUIM-DCs module.

• ink?

See also

• ink?

ink? Generic function

Returns #t if its argument is an ink.

Signature  ink? object => boolean

Parameters

• object – An instance of type <object>.

Values

• boolean – An instance of type <boolean>.

Discussion  Returns #t if object is an ink, otherwise returns #f.

See also

• <ink>

$magenta Constant

The usual definition of the color magenta.

Type  <color>

Discussion  The usual definition of the color magenta.

See also
make Generic function
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
Signature make (class == <brush>) #key foreground background mode fill-style fill-rule tile stipple ts-x ts-y => brush

Parameters
• (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: $boole-1.
  • fill-style – A (fill-style) or #f. Default value: #f.
  • fill-rule – A (fill-rule) or #f. Default value: #f.
  • tile – An (image) or #f. Default value: #f.
  • stipple – A (stipple) or #f. Default value: #f.
  • ts-x – An instance of false-or(<integer>). Default value: #f.
  • ts-y – An instance of false-or(<integer>). Default value: #f.

Values
• pen – An instance of type <pen>.
• brush – An instance of type <brush>.

Discussion 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
• <brush>
• <pen>

make-color-for-contrasting-color Generic function
Returns a color that is recognizably different from the main color.

Signature make-color-for-contrasting-color ink => color

Parameters
• ink – An instance of type <ink>.
Values
• **color** – An instance of type `<color>`.

Discussion Returns a color that is recognizably different from the main color.

See also
• **make-contrasting-colors**

**make-contrasting-colors Function**
Returns a vector of colors with recognizably different appearance.

Signature `make-contrasting-colors n #key k => colors`

Parameters
• `n` – An instance of type `<integer>`.
• `k` – An instance of type `<integer>`.
• `colors` – An instance of type `limited(<sequence>, of: <color>)`.

Discussion
Returns a vector of `n` colors with recognizably different appearance. Elements of the vector are guaranteed to be acceptable values for the `brush:` argument to the drawing functions, and do not include `$foreground`, `$background`, or `nil`. Their class is otherwise unspecified. The vector is a fresh object that may be modified.

If `k` is supplied, it must be an integer between 0 and `n - 1` (inclusive), in which case `make-contrasting-colors` returns the `k` th color in the vector rather than the whole vector.

If the implementation does not have `n` different contrasting colors, `make-contrasting-colors` signals an error. This does not happen unless `n` is greater than eight.

The rendering of the color is a true color or a stippled pattern, depending on whether the output medium supports color.

See also
• **contrasting-colors-limit**
• `$green`
• **make-color-for-contrasting-color**
• **make-contrasting-dash-patterns**

**make-contrasting-dash-patterns Function**
Returns a vector of dash patterns with recognizably different appearances.

Signature `make-contrasting-dash-patterns n #key k => dashes`

Parameters
• `n` – An instance of type `<integer>`.
• `k` – An instance of type `<integer>`.

Values
• `dashes` – An instance of type `<vector>`.

Discussion
Returns a vector of \( n \) dash patterns with recognizably different appearances. If the keyword \( k \) is supplied, `make-contrasting-dash-patterns` returns the \( k \)th pattern. If there are not \( n \) different dash patterns, an error is signalled.

The argument \( n \) represents the number of dash patterns.

The argument \( k \) represents the index in the vector of dash patterns indicating the pattern to use.

See also

- `contrasting-dash-patterns-limit`
- `make-contrasting-colors`

**make-device-font Function**

Returns a device-specific font.

**Signature**

```
make-device-font port font => device-font
```

**Parameters**

- `port` – An instance of type `<silica>`.
- `font` – An instance of type `<object>`.

**Values**

- `device-font` – A font object or the name of a font.

**Discussion**

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.

**make-gray-color Function**

Returns a member of class `<color>`.

**Signature**

```
make-gray-color luminosity #key opacity => color
```

**Parameters**

- `luminosity` – An instance of type `limited(<real>, min: 0, max: 1)`.
- `opacity` – An instance of type `limited(<real>, min: 0, max: 1)`. Default value: 1.0.

**Values**

- `color` – An instance of type `<color>`.

**Discussion**

Returns a member of class `<color>`. The *luminance* 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

- `make-ihs-color`
• make-rgb-color

make-ihs-color Function
Returns a member of the class <color>.

Signature  make-ihs-color intensity hue saturation #key opacity => color

Parameters

• intensity – An instance of type limited(<real>, min: 0, max: sqrt(3)).
• hue – An instance of type limited(<real>, min: 0, max: 1).
• saturation – An instance of type limited(<real>, min: 0, max: 1).
• opacity – An instance of type limited(<real>, min: 0, max: 1). Default value: 1.0.

Values

• color – An instance of type <color>.

Discussion  Returns a member of class <color>. The intensity argument is a real number between 0 and sqrt(3) (inclusive). The hue and saturation arguments are real numbers between 0 and 1 (inclusive).

See also

• make-gray-color
• make-rgb-color

make-palette Generic function
Returns a member of the class <palette>.

Signature  make-palette port #key => palette

Parameters

• port – An instance of type <silica>.

Values

• palette – An instance of type <palette>.

Discussion  Returns a member of the class <palette>.

make-pattern Function
Returns a pattern generated from a two-dimensional array.

Signature  make-pattern array colors => pattern

Parameters

• array – An instance of type <array>.
• colors – An instance of type limited(<sequence>, of: <color>).

Values

• pattern – An instance of type <pattern>.

Discussion  Returns a pattern design that has (array-dimension array 0) cells in the vertical direction and (array-dimension array 1) cells in the horizontal direction. array must be a two-dimensional array of non-negative integers less than the length of designs. designs must be a sequence of
designs. The design in cell *i,j* of the resulting pattern is the *n* th element of designs, if *n* is the value of (aref array *i* *j*). For example, array can be a bit-array and designs 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.

**make-rgb-color Function**

Returns a member of class <color>.

**Signature**

```scheme
make-rgb-color red green blue #key opacity => color
```

**Parameters**

- **red** – An instance of type limited(<real>, min: 0, max: 1).
- **green** – An instance of type limited(<real>, min: 0, max: 1).
- **blue** – An instance of type limited(<real>, min: 0, max: 1).
- **opacity** – An instance of type limited(<real>, min: 0, max: 1). Default value: 1.0.

**Values**

- **color** – An instance of type <color>.

**Discussion**

Returns a member of class <color>. The red, green, and blue 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**

- make-gray-color
- make-ihs-color

**make-stencil Function**

Returns a pattern design generated from a two-dimensional array.

**Signature**

```scheme
make-stencil array => stencil
```

**Parameters**

- **array** – An instance of type <array>.

**Values**
• **stencil** – An instance of type `<stencil>`.

**Discussion**  Returns a pattern design that has \((array\text{-}dimension\ array\ 0)\) cells in the vertical direction and \((array\text{-}dimension\ array\ 1)\) cells in the horizontal direction. \(array\) must be a two-dimensional array of real numbers between 0 and 1 (inclusive) that represent opacities. The design in cell \(i,j\) of the resulting pattern is the value of \((\text{make\text{-}opacity} (\text{aref} array i j))\).

### `make-text-style` Function

**Returns** an instance of `<text-style>`.

**Signature**  `make-text-style` `family` `weight` `slant` `size` `#key` `underline?` `strikeout?` `=>` `text-style`

**Parameters**

- `family` – An instance of type `one-of(#"fix", #"serif", #"sans-serif", #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)`.
- `underline?` – An instance of type `<boolean>`.
- `strikeout?` – An instance of type `<boolean>`.

**Values**

- `text-style` – An instance of type `<text-style>`.

**Discussion**  Returns an instance of `<text-style>`.

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 \#f in them) may be filled in by the components of a default style object. A text style object is called **fully specified** if none of its components is \#f, and the size component is not a relative size (that is, neither #"smaller" nor #"larger").

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**

- `$solid-pen`

### `merge-text-styles` Generic function

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**  `merge-text-styles` `text-style` `default-style` `=>` `text-style`

**Parameters**

- `text-style` – An instance of type `<text-style>`.
- `default-style` – An instance of type `<text-style>`.
Values

• **text-style** – An instance of type `<text-style>`.

Discussion

Merges the text styles `text-style` with `default-style`, that is, returns a new text style that is the same as `text-style`, except that unspecified components in style1 are filled in from `default-style`. For convenience, the two arguments may be also be style specs. Note that `default-style` must be a fully specified text style.

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 `default-style`. The ordering of sizes, from smallest to largest, is `"tiny"`, `"very-small"`, `"small"`, `"normal"`, `"large"`, `"very-large"`, and `"huge"`.

See also

• `default-background-setter`

`<palette>` Abstract Instantiable Class

The protocol class for color palettes.

**Superclasses** `<object>`

**Discussion** The protocol class for color palettes.

**Operations**

• `add-colors`
• `do-add-colors`
• `remove-colors`
• `do-remove-colors`
• `color-palette?`
• `dynamic-palette?`

See also

• `palette?`

`palette?` Generic function

Returns `#t` if an object is a palette.

**Signature** `palette? object => boolean`

**Parameters**

• `object` – An instance of type `<object>`.

**Values**

• `boolean` – An instance of type `<boolean>`.

**Discussion** Returns `#t` if the object `object` is a palette. A palette is a color map that maps 16 bit colors into a, for example, 8 bit display.

See also

• `<palette>`

`<palette-full>` Concrete Sealed Class

The class for errors that are signalled when a color palette is full.
Superclasses  <error>

Init-Keywords
  • palette –

Discussion  The class for errors that are signalled when a color palette is full.

See also
  • <palette>

$parquet-stipple Constant
A stipple pattern for use in creating a patterned brush that looks like a parquet floor.

Type  <array>

Discussion  A stipple pattern for use in creating a patterned brush that looks like a parquet floor.

See also
  • brush-stipple

<pattern> Concrete Sealed Class
The class for patterns.

Superclasses  <stencil>

Init-Keywords
  • colors – An instance of type limited(<sequence>, of:  <color>).

Discussion  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 DUIM-DCs module.
  • pattern?

See also
  • <stencil>
  • make-pattern

pattern? Generic function
Returns #t if its argument is a pattern.

Signature  pattern?  object =>  boolean

Parameters
  • object – An instance of type <object>.

Values
  • boolean – An instance of type <boolean>.

Discussion  Returns #t if object is a pattern.

See also
  • make-pattern

<pen> Abstract Instantiable Class
The protocol class for pens.

Superclasses  <object>
Init-Keywords

- **width** – An instance of type `<integer>`. Default value: 1.
- **units** – An instance of type one-of(#"normal", #"point", #"device"). Default value: #"normal".
- **dashes** – An instance of type-union(<boolean>, <sequence>). Default value: #f.
- **joint-shape** – An instance of type one-of(#"miter", #"bevel", #"round", #"none"). Default value: #"miter".
- **cap-shape** – An instance of type one-of(#"butt", #"square", #"round", #"no-end-point"). Default value: #"butt".

Discussion  The protocol class for pens. A pen imparts ink to a medium.

Operations

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

- =
- `pen?`
- `pen-cap-shape`
- `pen-dashes`
- `pen-joint-shape`
- `pen-units`
- `pen-width`

See also

- `<ink>`
- `make`
- `pen?`
- `pen-cap-shape`
- `pen-dashes`
- `pen-joint-shape`
- `pen-units`
- `pen-width`

`pen?`  Generic function

Returns #t if its argument is a pen.

Signature  `pen? object => boolean`

Parameters

- **object** – An instance of type `<object>`.

Values

- **boolean** – An instance of type `<boolean>`.

Discussion  Returns #t if `object` is a pen, otherwise returns #f.

See also
pen-cap-shape Generic function
Returns the shape of the end of a line or an arc drawn by the pen.

Signature  pen-cap-shape pen => value
Parameters
  • pen – An instance of type <pen>.
Values
  • value – An instance of type one-of(#"butt", #"square", #"round", #"no-end-point").
Discussion  Returns the shape of the end of a line or an arc drawn by pen.
See also
  • make-contrasting-dash-patterns
  • <pen>
  • pen?
  • pen-dashes
  • pen-joint-shape
  • pen-units
  • pen-width

pen-dashes Generic function
Returns #t if the lines drawn by a pen are dashed.

Signature  pen-dashes pen => value
Parameters
  • pen – An instance of type <pen>.
Values
  • value – An instance of type type-union(<boolean>, <sequence>).
Discussion  Returns #t if the lines drawn by pen 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
  • <pen>
  • pen?
  • pen-cap-shape
• *pen-joint-shape*
  • *pen-units*
  • *pen-width*

**pen-joint-shape** Generic function

Returns the shape of the joints between line segments of a closed, unfilled figure.

**Signature**  \texttt{pen-joint-shape pen \Rightarrow value}

**Parameters**

• \texttt{pen} – An instance of type \texttt{<pen>}.

• \texttt{value} – An instance of type \texttt{one-of(#"miter", #"bevel", #"round", #"none")}.

**Discussion** Returns the shape of the joints between line segments of a closed, unfilled figure drawn by \texttt{pen}.

See also

• *make-contrasting-dash-patterns*
  • *<pen>*
  • *pen?*
  • *pen-cap-shape*
  • *pen-dashes*
  • *pen-units*
  • *pen-width*

**pen-units** Generic function

Returns the units in which the pen width is specified.

**Signature**  \texttt{pen-units pen \Rightarrow value}

**Parameters**

• \texttt{pen} – An instance of type \texttt{<pen>}.

**Values**

• \texttt{value} – An instance of type \texttt{one-of(#"normal", #"point", #"device")}.

**Discussion** Returns the units in which the pen width is specified. They may be normal, points, or device-dependent. A width of \texttt{#"normal"} is a comfortably visible thin line.

See also

• *make-contrasting-dash-patterns*
  • *<pen>*
  • *pen?*
  • *pen-cap-shape*
  • *pen-dashes*
  • *pen-joint-shape*
  • *pen-width*
pen-width Generic function
Returns the pen-width, that is how wide a stroke the pen draws, of its argument.

Signature  pen-width pen => width

Parameters
• pen – An instance of type <pen>.

Values
• width – An instance of type <pen-width>. The units that specify the width of the pen may be "normal", "points", or "device".

Discussion Returns the pen width, that is how wide a stroke the pen draws, of pen. A width of "normal" is a comfortably visible thin line.

See also
• make-contrasting-dash-patterns
• <pen>
• pen?
• pen-cap-shape
• pen-dashes
• pen-joint-shape
• pen-units

read-image Generic function
Reads an image.

Signature  read-image resource-id #key image-type: image-type #all-keys => image

Parameters
• locator – An instance of type type-union(<string>, <locator>).

• image-type – On Windows, an instance of type one-of(#"bitmap", #"icon").

Values
• image – An instance of type <image>.

Discussion Reads an image from the location resource-id. This function calls read-image-as.

See also
• read-image-as

read-image-as Generic function
Reads an image.

Signature  read-image-as class locator image-type #key #all-keys => image

Parameters
• class – An instance of type <object>.

• locator – An instance of type <string>.

• image-type – On Windows, #"bitmap" or #"icon".

Values
• image – An instance of type <image>.
Discussion

Reads the image in the location pointed to be locator, as an instance of a particular class*. This function is called by read-image.

The class represents the class that the image is read as an instance of.

See also

• read-image

$red Constant

The usual definition of the color red.

Type <color>

Discussion The usual definition of the color red.

See also

• $blue

remove-colors Generic function

Removes one or more colors from a palette and returns the updated palette.

Signature remove-colors palette #rest colors => palette

Parameters

• palette – An instance of type <palette>.
• colors – Instances of type <color>.

Values

• palette –

Discussion Removes colors from palette and returns the updated palette.

$solid-pen Constant

A pen that draws a solid line.

Type <pen>

Discussion A pen that draws a solid line. The width of the line is 1, and dashes: is #f.

See also

• <pen>
• make
• $dash-dot-pen
• $dotted-pen

<stencil> Concrete Sealed Class

The class for stencils.

Superclasses <image>

Init-Keywords

• array – An instance of type <array>. Required.
• transform – An instance of type <transform>. Default value: #f.

Discussion The class for stencils. A stencil is a special kind of pattern that contains only opacities.
Operations

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

- `image-height`
- `image-width`
- `stencil?`

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

- `box-edges`

See also

- `<image>`
- `make-pattern`
- `stencil?`

**stencil?** Generic function

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

**Signature**  
`sentence stencil? object => boolean`

**Parameters**

- `object` – An instance of type `<object>`.

**Values**

- `boolean` – An instance of type `<boolean>`.

**Discussion**  
Returns `#t` if its argument is a stencil.

See also

- `make-pattern`
- `<stencil>`

**<text-style>** Abstract Instantiable Class

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`.
- `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`.
Discussion

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

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

- =
- 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

The following operations are exported from the DUIM-Sheets module.

- medium-default-text-style
- medium-default-text-style-setter
- medium-merged-text-style
- medium-text-style
- medium-text-style-setter

See also

- text-style?
- text-style-components
- text-style-family
- text-style-size
- text-style-slant
- text-style-strikeout?
- text-style-underline?
- text-style-weight
text-style? Generic function
Returns #t if its argument is a text-style.

Signature  text-style? object => text-style?

Parameters

• object – An instance of type <object>.

Values

• text-style? – An instance of type <boolean>.

Discussion  Returns #t if its argument is a text-style.

See also

• <text-style>
• text-style-components
• text-style-family
• text-style-size
• text-style-slant
• text-style-strikeout?
• text-style-underline?
• text-style-weight

text-style-components Generic function
Returns the components of a text style as the values family, face, slant, size, underline and strikeout.

Signature  text-style-components text-style => family weight slant size underline? strikeout?

Parameters

• text-style – An instance of type <text-style>.
• slant – An instance of type one-of(#"roman", #"italic", #"oblique", #f).

Values

• family – An instance of type one-of(#"fix", #"serif", #"sans-serif", #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>.

• strikeout? – An instance of type <boolean>.

Discussion  Returns the components of the text style text-style as the values family, face, slant, size, underline and strikeout.

See also

• <text-style>
• text-style?
• text-style-family
• text-style-size
• text-style-slant
• text-style-strikeout?
• text-style-underline?
• text-style-weight

text-style-family Generic function
Returns the family component of the specified text style.

Signature text-style-family text-style => family

Parameters
• text-style – An instance of type <text-style>.

Values
• family – An instance of type one-of(#"fix", #"serif", #"sans-serif", #f).

Discussion Returns the family component of the specified text style.

See also
• <text-style>
• text-style?
• text-style-components
• text-style-size
• text-style-slant
• text-style-strikeout?
• text-style-underline?
• text-style-weight

text-style-size Generic function
Returns the style component of the specified text style.

Signature text-style-size text-style => size

Parameters
• text-style – An instance of type <text-style>.

Values
• 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.

Discussion Returns the style component of the specified text style.

See also
• <text-style>
• text-style?
• \textstyle-components
• \textstyle-family
• \textstyle-slant
• \textstyle-strikeout?
• \textstyle-underline?
• \textstyle-weight

\textstyle-slant \textbf{Generic function}
\textit{Returns the slant component of the specified text style.}

\textbf{Signature} \textstyle-slant \textstyle => slant

\textbf{Parameters}
• \textstyle – An instance of type \textstyle.

\textbf{Values}
• slant – An instance of type one-of(#"roman", #"italic", #"oblique", #f).

\textbf{Discussion} Returns the slant component of the specified text style.

\textbf{See also}
• \textstyle
• \textstyle?
• \textstyle-components
• \textstyle-family
• \textstyle-size
• \textstyle-strikeout?
• \textstyle-underline?
• \textstyle-weight

\textstyle-strikeout? \textbf{Generic function}
\textit{Returns #t if the text style includes a line through it, striking it out.}

\textbf{Signature} \textstyle-strikeout? \textstyle => strikeout?

\textbf{Parameters}
• \textstyle – An instance of type \textstyle.

\textbf{Values}
• strikeout? – An instance of type boolean.

\textbf{Discussion} Returns #t if the text style includes a line through it, striking it out.

\textbf{See also}
• \textstyle
• \textstyle?
• \textstyle-components
• \textstyle-family
• text-style-size
• text-style-slant
• text-style-underline?
• text-style-weight

**text-style-underline?** Generic function
Returns #t if the text style is underlined.

**Signature**  text-style-underline?  text-style => underline?

**Parameters**
• text-style – An instance of type `<text-style>`.

**Values**
• underline? – An instance of type `<boolean>`.

**Discussion**  Returns #t if the text style is underlined.

**See also**
• `<text-style>`
• text-style?
• text-style-components
• text-style-family
• text-style-size
• text-style-slant
• text-style-strikeout?
• text-style-weight

**text-style-weight** Generic function
Returns the weight component of the specified text style.

**Signature**  text-style-weight  text-style => weight

**Parameters**
• text-style – An instance of type `<text-style>`.

**Values**

**Discussion**  Returns the weight component of the text style.

**See also**
• `<text-style>`
• text-style?
• text-style-components
• text-style-family
• text-style-size
$tiles-stipple Constant

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

Type <array>

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

See also

- brush-stipple

$vertical-hatch Constant

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

Type <array>

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

See also

- brush-stipple

$white Constant

The usual definition of white.

Type <color>

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

See also

- <color>

write-image Generic function

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

Signature write-image image locator => ()

Parameters

- image – An instance of type <image>.
- locator – An instance of type <string>.

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

$xor-brush Constant

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

Type <brush>

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

$yellow Constant

The usual definition of the color yellow.

Type <color>
Discussion  The usual definition of the color yellow.

See also

- `<color>`
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. `DUIM-Sheets Module` 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 repaint 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 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
in the Open Dylan installation directory.

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.

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 the following table. Only `<sheet>`, and `<event>` have any subclasses defined. An `<event>` is an object representing some sort of event. See Subclasses of `<event>` for details of the subclasses of `<event>`.

Overall class hierarchy for the DUIM-Sheets library

<table>
<thead>
<tr>
<th><code>&lt;object&gt;</code></th>
<th><code>&lt;sheet&gt;</code></th>
<th><code>&lt;display&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;port&gt;</code></td>
<td><code>&lt;clipboard&gt;</code></td>
<td><code>&lt;caret&gt;</code></td>
</tr>
<tr>
<td><code>&lt;pointer&gt;</code></td>
<td><code>&lt;medium&gt;</code></td>
<td><code>&lt;frame-manager&gt;</code></td>
</tr>
<tr>
<td><code>&lt;event&gt;</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `<sheet>` 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 — `<display>` — which is an object that represents a single display (or screen) on a display server. All sheets can be attached to a display.
- `<port>` 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 Subclasses of `<device-event>`.

- `<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 Subclasses of `<device-event>`.

- `<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 using the default frame manager.

### Subclasses of `<event>`

The following table shows the subclasses of the `<event>` class that are exposed by the DUIM-Sheets library.

<table>
<thead>
<tr>
<th><code>&lt;event&gt;</code></th>
<th>Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;frame-event&gt;</code></td>
<td><code>&lt;port-terminated-event&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;timer-event&gt;</code></td>
</tr>
<tr>
<td><code>&lt;sheet-event&gt;</code></td>
<td><code>&lt;device-event&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;window-event&gt;</code></td>
</tr>
<tr>
<td></td>
<td>See Subclasses of <code>&lt;device-event&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;window-configuration-event&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;window-repaint-event&gt;</code></td>
</tr>
</tbody>
</table>

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 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 Subclasses of `<device-event>`.
- 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.

### Subclasses of `<device-event>`

The following table 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>
<thead>
<tr>
<th><code>&lt;device-event&gt;</code></th>
<th><code>&lt;pointer-event&gt;</code></th>
<th><code>&lt;pointer-button-event&gt;</code></th>
<th><code>&lt;button-press-event&gt;</code></th>
<th><code>&lt;button-release-event&gt;</code></th>
<th><code>&lt;button-click-event&gt;</code></th>
<th><code>&lt;double-click-event&gt;</code></th>
<th><code>&lt;pointer-drag-event&gt;</code></th>
<th><code>&lt;pointer-motion-event&gt;</code></th>
<th><code>&lt;pointer-exit-event&gt;</code></th>
<th><code>&lt;pointer-enter-event&gt;</code></th>
<th><code>&lt;key-press-event&gt;</code></th>
<th><code>&lt;key-release-event&gt;</code></th>
</tr>
</thead>
</table>

**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 The base classes in the DUIM-Sheets library.

- `<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:
DUIM Reference Documentation, Release 1.0

- `<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.

**DUIM-Sheets Module**

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

---

1. **Generic function**
   - Returns true if the specified gestures are the same.

   **Signature**
   
   = gesture1 gesture2 => equal?

   **Parameters**
   
   - gesture1 – An instance of type `<gesture>`.
   - gesture2 – An instance of type `<gesture>`.

   **Values**
   
   - equal? – An instance of type `<boolean>`.

   **Discussion** Returns true if gesture1 and gesture2 are the same.

   **See also**
   
   - *gesture-spec-equal

2. **add-child**
   - Generic function
   
   Adds a child to the specified sheet.

   **Signature**
   
   add-child sheet child #key index => sheet

   **Parameters**
   
   - sheet – An instance of type `<sheet>`.
   - child – An instance of type `<sheet>`.
   - index – An instance of type `false-or(<integer>)`.

   **Values**
   
   - sheet – An instance of type `<sheet>`.

   **Discussion** Adds a child to sheet.

   **See also**
   
   - *remove-child
add-clipboard-data Generic function
Adds data to a clipboard.

**Signature**

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

**Parameters**

- `clipboard` – An instance of `<clipboard>`.
- `data` – An instance of `<object>`.

**Values**


**Discussion**

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

add-clipboard-data-as Generic function
Coerces data to a particular type and then adds it to a clipboard.

**Signature**

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

**Parameters**

- `type` – An instance of `type-union(<symbol>, <type>)`.
- `clipboard` – An instance of `<clipboard>`.
- `data` – An instance of `<object>`.

**Values**


**Discussion**

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

$alt-key Constant

A constant that represents the ALT key on the keyboard.

**Type** `<integer>`

**Value** `$meta-key`

**Discussion**

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

- `$control-key`
- `$hyper-key`
- `$meta-key`
- `modifier-key-index`
- `modifier-key-index-name`
- `$modifier-keys`
- `$option-key`
- `$shift-key`

6.3. DUIM-Sheets Module
• $super-key

**beep** Generic function

**Signature**  
`beep drawable => ()`

**Parameters**

- `drawable` – An instance of type `type-union(<sheet>, <medium>)`.

**Discussion**

**boundary-event-kind** Generic function

Returns the kind of boundary event for the specified event.

**Signature**  
`boundary-event-kind event => symbol`

**Parameters**

- `event` – An instance of type `<event>`.

**Values**


**Discussion**  
Returns the kind of boundary event for `event`. These correspond to the detail members for X11 enter and exit events.

See also

- `<pointer-boundary-event>`

**button-index** Function

Returns the index for the specified pointer button.

**Signature**  
`button-index button => index`

**Parameters**

- `button` – An instance of type `one-of(#"left", #"middle", #"right")`.

**Values**

- `index` – An instance of type `<integer>`.

**Discussion**  
Returns the index for `button`, a button on the pointer device connected to the computer (typically a mouse). The `index` returned is either 0, 1, or 2, for the left, middle, or right buttons, respectively.

See also

- `button-index-name`
- `$pointer-buttons`

**button-index-name** Function

Returns the button on the pointer device represented by the specified index.

**Signature**  
`button-index-name index => button`

**Parameters**

- `index` – An instance of type `<integer>`.

**Values**

- `button` – An instance of type `one-of(#"left", #"middle", #"right")`. 
**Discussion** Returns the button on the pointer device connected to the computer (typically a mouse) represented by `index`. The `index` is either 0, 1, or 2, these values corresponding to the left, middle, or right buttons, respectively.

See also
- `button-index`
- `$pointer-buttons`

**<button-press-event> Instantiable Sealed Class**
The class of events representing button presses.

**Superclasses** `<pointer-button-event>`

**Discussion** 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.

See also
- `<button-release-event>`
- `<double-click-event>`

**<button-release-event> Instantiable Sealed Class**
The class of events representing button releases.

**Superclasses** `<pointer-button-event>`

**Discussion** 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.

See also
- `<button-press-event>`

**<caret> Abstract Instantiable Class**
The class of carets.

**Superclasses** `<object>`

**Init-Keywords** `<object>`

**Discussion**
The class of carets, or text cursors. A cursor can actually be any instance of `<symbol>` or any instance of `<image>`.

The `sheet`: init-keyword specifies the sheet that the caret is positioned in.

The `x`, `y`, `width`, and `height`: 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.
Operations

- caret-position
- caret-sheet
- caret-size
- caret-visible?
- caret-visible?-setter
- display
- port
- set-caret-position

See also

- caret-position
- caret-sheet
- caret-size
- caret-visible?
- <cursor>

caret-position Generic function

Returns the position of the specified caret.

Signature  cursor-position caret => x y

Parameters

- caret – An instance of type <caret>.

Values

- x – An instance of type <integer>.
- y – An instance of type <integer>.

Discussion  Returns the position of caret.

See also

- caret-sheet
- caret-size

caret-sheet Generic function

Returns the sheet that owns the specified caret.

Signature  cursor-sheet caret => sheet

Parameters

- caret – An instance of type <caret>.

Values

- sheet – An instance of type <sheet>.

Discussion  Returns the sheet that owns caret.

See also

- caret-position
• caret-size

caret-size Generic function
Returns the size of the specified caret.

Signature  cursor-size caret => width height

Parameters
• caret – An instance of type <caret>.

Values
• width – An instance of type <integer>.
• height – An instance of type <integer>.

Discussion  Returns the size of caret.

See also
• caret-position
• caret-sheet

caret-visible? Generic function
Returns true if the specified caret is visible.

Signature  cursor-visible? caret => visible?

Parameters
• caret – An instance of type <caret>.

Values
• visible? – An instance of type <boolean>.

Discussion  Returns true if caret is visible.

See also
• <cursor>
• caret-visible?-setter

caret-visible?-setter Generic function
Specifies whether or not the specified caret is visible.

Signature  cursor-visible?-setter visible? caret => boolean

Parameters
• visible? – An instance of type <boolean>.
• caret – An instance of type <caret>.

Values
• boolean – An instance of type <boolean>.

Discussion  Specifies whether or not caret is visible.

See also
• <cursor>
• caret-visible?
**child-containing-position** Generic function

Returns the topmost child of the specified sheet that occupies a specified position.

**Signature**

```
child-containing-position sheet x y => value
```

**Parameters**

- **sheet** – An instance of type `<sheet>`.
- **x** – An instance of type `<real>`.
- **y** – An instance of type `<real>`.

**Values**

- **value** – An instance of type `false-or(<sheet>)`.

**Discussion**

Returns the topmost enabled direct child of `sheet` whose region contains the position `(x, y)`. The position is expressed in the coordinate system used by `sheet`.

**See also**

- `children-overlapping-region`
- `do-child-containing-position`

**children-overlapping-region** Generic function

Returns any children of the specified sheet whose regions overlap a specified region.

**Signature**

```
children-overlapping-region sheet region => sheets
```

**Parameters**

- **sheet** – An instance of type `<sheet>`.
- **region** – An instance of type `<region>`.

**Values**

- **sheets** – An instance of type `limited(<sequence>, of: <sheet>)`.

**Discussion**

Returns the list of enabled direct children of `sheet` whose region overlaps `region`.

**See also**

- `child-containing-position`
- `do-children-overlapping-region`

**choose-color** Generic function

Displays the built-in color dialog for the target platform.

**Signature**

```
choose-color #key frame owner title documentation exit-boxes name default => color
```

**Parameters**

- **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>`

Discussion

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.

![Choose Color Dialog](image)

Fig. 6.1: The standard Choose Color 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.

**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 Generic function
Displays the built-in directory dialog for the target platform.

Signature  choose-directory #key frame owner title documentation exit-boxes name default => locator

Parameters
• 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
• locator – An instance of type type-union(<string>, <locator>).

Discussion
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.

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.

Example 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.

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

Generic function
Displays the built-in file dialog for the target platform.

**Signature**
choose-file #key frame owner title documentation exit-boxes name default => locator

**Parameters**
- **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>.
- **direction** – An instance of type one-of(#"input", #"output"). Default value: #"input".
- **filters** – An instance of type limited(<sequence>, of: <sequence>).
- **exit-boxes** – An instance of type <object>.
- **name** – An instance of type <object>.
- **default** – An instance of type <string>.

**Values**
- **locator** – An instance of type <string>.

**Discussion**
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.

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:

1. The first string in the sequence is a description of the files that are displayed when this filter is chosen.
2. 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:

```lisp
#"["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.

```lisp
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
• choose-directory
• notify-user

choose-from-dialog Generic function
Prompt the user to choose from a collection of items, using a dialog box.

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

Parameters
• items – An instance of type-union(<sequence>, <menu>).
• frame – An instance of type <frame>. Default value: #f.
• owner – An instance of type <sheet>. Default value: #f.
• title – An instance of type <string>.
• default-item – An instance of type <object>.
• label-key – An instance of type <function>. Default value: identity.
• value-key – An instance of type <function>. Default value: identity.
• selection-mode – An instance of <symbol>. Default value: #"single".
• gadget-class – An instance of type <gadget>.
• gadget-options – An instance of type <sequence>.
• foreground – An instance of type <ink>.
• background – An instance of type <ink>.
• text-style – An instance of type <text-style>.
Values

- **value** – An instance of type `<object>.
- **success?** – An instance of type `<boolean>.

Discussion

Prompt the user to choose from a collection of *items*, using a dialog box. This generic function is similar to `choose-from-menu`.

The function returns the values chosen by the user, and a boolean value: `#t` if a value was chosen, `#f` if nothing was chosen. Unlike `choose-from-menu`, the user can choose several values if desired, depending on the value of `selection-mode`, described below.

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

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

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

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

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

See also

- `choose-from-menu`

choose-from-menu Generic function

Prompt the user to choose from a collection of items, using a pop-up menu.

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

**Parameters**

- **items** – An instance of type-union(<sequence>, <menu>.
- **frame** – An instance of type `<frame>`. Default value: `#f`.
- **owner** – An instance of type `<sheet>`. Default value: `#f`.
- **title** – An instance of type `<string>`. Default value: `#f`.
- **default-item** – An instance of type `<object>.

132 Chapter 6. DUIM-Sheets Library
• **label-key** – An instance of type `<function>`. Default value: `identity`.
• **value-key** – An instance of type `<function>`. Default value: `identity`.
• **foreground** – An instance of type `<ink>`.
• **background** – An instance of type `<ink>`.
• **text-style** – An instance of type `<text-style>`.

**Values**

• **value** – An instance of type `<object>`.
• **success?** – An instance of type `<boolean>`.

**Discussion**

Prompt the user to choose from a collection of *items*, using a pop-up menu. This generic function is similar to `choose-from-dialog`.

The function returns the value chosen by the user, and a boolean value: `#t` if a value was chosen, `#f` if nothing was chosen.

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

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

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

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

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

See also

• `choose-from-dialog`

**choose-text-style** Generic function

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

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

**Parameters**

• **frame** – An instance of type `<frame>`. Default value: `#f`.
• **owner** – An instance of type `<sheet>`. Default value: `#f`.
• **title** – An instance of type `<string>`. Default value: `#f`.

**Values**

• **font** – An instance of `<text-style>`.
Discussion

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

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.

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

clear-box Generic function
Cleareds a box-shaped area in the specified drawable.

Signature clear-box drawable left top right bottom => ()
Signature clear-box* drawable region => ()
Parameters
  • drawable – An instance of type type-union(<sheet>, <medium>).

The following arguments are specific to clear-box.

Parameters
  • left – An instance of type <coordinate>.
  • top – An instance of type <coordinate>.
  • right – An instance of type <coordinate>.
  • bottom – An instance of type <coordinate>.

The following argument is specific to clear-box*.

Parameters
  • region – An instance of type <region>.

Discussion
Cleareds 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
Cleareds the contents of a clipboard.

Signature clear-clipboard clipboard => ()
Parameters
  • clipboard – An instance of <clipboard>.

Discussion Cles the contents of clipboard, which represents the locked clipboard.

<clipboard> Open Abstract Class
The class of clipboard objects.

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

See also

- `add-clipboard-data`
- `add-clipboard-data-as`
- `clear-clipboard`
- `clipboard-data-available?`
- `clipboard-sheet`
- `clipboard-owner`
- `close-clipboard`
- `get-clipboard-data-as`
- `open-clipboard`
- `with-clipboard`

**clipboard-data-available?** Generic function

Returns false if there is any data of a particular type on a clipboard.

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

**Parameters**

- **type** – An instance of `type-union(<symbol>, <type>)`.
- **clipboard** – An instance of `<clipboard>`.

**Values**

- **available?** – An instance of `<boolean>`.

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

See also

- `add-clipboard-data`
- `add-clipboard-data-as`
- `<clipboard>`
- `get-clipboard-data-as`

**clipboard-sheet** Generic function

Returns the sheet with the clipboard lock.

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

**Parameters**

- **clipboard** – An instance of `<clipboard>`.

**Values**

- **sheet** – An instance of `<sheet>`.

**Discussion**  Returns the sheet with the clipboard lock.

See also
• <clipboard>

**clipboard-owner** Generic function

Returns the sheet that owns the current clipboard data.

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

**Parameters**

- `clipboard` – An instance of `<clipboard>`.

**Values**

- `owner` – An instance of `<sheet>`.

**Discussion**

Returns the sheet that owns the current clipboard data.

See also

• `<clipboard>`

**close-clipboard** Function

Closes the current clipboard lock for a sheet on a port.

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

**Parameters**

- `port` – An instance of `<port>`.
- `sheet` – An instance of `<sheet>`.

**Discussion**

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

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

See also

• `<clipboard>`
- `with-clipboard`

**$control-key** Constant

A constant that represents the CONTROL key on the keyboard.

**Type** `<integer>`

**Value**  
`ash(1, %modifier_base + 1);`

**Discussion**

A constant that represents the CONTROL key on the keyboard.

See also

• `$alt-key`
- `$hyper-key`
- `$meta-key`
- `modifier-key-index`
- `modifier-key-index-name`
- `$modifier-keys`
- `$option-key`
• $shift-key
• $super-key

cursor> Class
The class of cursor objects.
Equivalent: type-union(<symbol>, <image>)

Discussion 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 <symbol> or any instance of <image>.

Operations
• pointer-cursor-setter
• set-caret-position
• sheet-pointer-cursor-setter

See also
• <caret>
• cursor?

cursor? Generic function
Returns true if the specified object is a cursor.

Signature cursor? object => cursor?

Parameters
• object – An instance of type <object>.

Values
• cursor? – An instance of type <boolean>.

Discussion Returns true if object is a cursor. In practice, you can create a cursor from any instance of <symbol> or <image>.

See also
• <cursor>

default-port Function
Returns the default port for the specified server.

Signature default-port #key server-path => port

Parameters
• server-path – An instance of type <vector>. Default value: #("local").
• port – An instance of type false-or(<port>).

Discussion Returns the default port for server specified by server-path.

See also
• default-port-setter
• destroy-port

default-port-setter Function
Sets the default port.
Signature default-port-setter port => port

Parameters

- **port** – An instance of type `<port>`. Default value: `#f`.

Values

- **port** – An instance of type `<port>`.

Discussion Sets the default port.

See also

- `default-port`
- `destroy-port`

**destroy-port** Generic function

Destroys the specified port.

Signature destroy-port port => ()

Parameters

- **port** – An instance of type `<port>`.

Discussion Destroys `port`.

See also

- `default-port`
- `default-port-setter`

**destroy-sheet** Generic function

Destroys the specified sheet.

Signature destroy-sheet sheet => ()

Parameters

- **sheet** – An instance of type `<sheet>`.

Discussion Destroys `sheet`.

**<device-event>** Open Abstract Class

The class of device events.

Superclasses `<sheet-event>`

Init-Keywords

- **sheet** – An instance of type `<sheet>`.
- **modifier-state** – An instance of type `<integer>`. Default value: 0.

Discussion

The class of device events.

The `modifier-state` init-keyword is used to record the state of the device at the time the event occurred.

Operations

- `event-modifier-state`

**<display>** Open Abstract Class

The class of displays.
Superclasses  <sheet>

Init-Keywords

- **orientation** – An instance of type `one-of(#"vertical", #"horizontal", #"default")`. Default value: #"default".
- **units** – An instance of type `one-of(#"device", #"mm", #"pixels")`. Default value: #"device".

Discussion

The class of displays. An instance of `<display>` is an object that represents a single display (or screen) on some display server. Any sheet can be attached to an instance of `<display>`, and a display, and all the sheets attached to it, are associated with a `<port>` that is a connection to a display server.

The **orientation**: init-keyword is used to specify the orientation of a display.

The **units**: 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).

Operations

- **display**
- **display?**
- **display-depth**
- **display-height**
- **display-mm-height**
- **display-mm-width**
- **display-orientation**
- **display-pixel-height**
- **display-pixels-per-point**
- **display-pixel-width**
- **display-units**
- **display-width**

See also

- **display**
- **display?**
- **display-depth**
- **display-height**
- **display-orientation**
- **display-units**
- **display-width**
- `<port>`
- `<sheet>`
display  Generic function
    Returns the display for the specified object.
    Signature   display  object  =>  display
    Parameters
        •      object  –  An instance of type  <object>.
        •      display  –  An instance of type  false-or(<display>).
    Discussion  Returns the display used to display  object.
    See also
        •  <display>
        •  frame-manager
        •  port

display?  Generic function
    Returns true if the specified object is a display.
    Signature   display?  object  =>  display?
    Parameters
        •      object  –  An instance of type  <object>.
    Values
        •      display?  –  An instance of type  <boolean>.
    Discussion  Returns true if  object  is a display.
    See also
        •  <display>

display-depth  Generic function
    Returns the color depth of the specified display.
    Signature   display-depth  display  =>  depth
    Parameters
        •      display  –  An instance of type  <display>.
    Values
        •      depth  –  An instance of type  <integer>.
    Discussion  Returns the color depth of  display. By default, the color depth of any display is assumed to be 8.
    See also
        •  display-height
        •  display-orientation
        •  display-width

display-height  Generic function
    Returns the height of the specified display.
    Signature   display-height  display  #key  units  =>  height
    Parameters
• display – An instance of type <display>.
• units – An instance of one-of(#"device", #"mm", #"pixels"). Default value: #"device".

Values
• height – An instance of type <number>.

Discussion Returns the height of display, in device-independent units. If units is specified, then the value returned is converted into the appropriate type of units.

See also
• display-depth
• display-mm-height
• display-orientation
• display-pixel-height
• display-units
• display-width

**display-mm-height Generic function**
Returns the height of the specified display in millimeters.

**Signature** display-mm-height display => height

**Parameters**
• display – An instance of type <display>.

**Values**
• height – An instance of type <number>.

**Discussion** Returns the height of display in millimeters. This is equivalent to calling display-height with the units argument set to #"mm".

See also
• display-height
• display-mm-width
• display-pixel-height
• display-units

**display-mm-width Generic function**
Returns the width of the specified display in millimeters.

**Signature** display-mm-width display => width

**Parameters**
• display – An instance of type <display>.

**Values**
• width – An instance of type <number>.

**Discussion** Returns the width of display in millimeters. This is equivalent to calling display-width with the units argument set to #"mm".

See also
• display-mm-height
• display-pixel-width
• display-units
• display-width

display-orientation Generic function
Returns the orientation of the specified display.

Signature  display-orientation display => orientation

Parameters
• display – An instance of type <display>.

Values
• orientation – An instance of type one-of(#"vertical", #"horizontal", #"default").

Discussion  Returns the orientation of display. Unless specified otherwise, the orientation of any display is #"default".

See also
• display-depth
• display-height
• display-width

display-pixel-height Generic function
Returns the height of the specified display in pixels.

Signature  display-pixel-height display => height

Parameters
• display – An instance of type <display>.

Values
• height – An instance of type <integer>.

Discussion  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
• display-mm-height
• display-pixel-width
• display-units

display-pixels-per-point Generic function
Returns the number of pixels per point for the specified display.

Signature  display-pixels-per-point display => number

Parameters
• display – An instance of type <display>.

Values
DUIM Reference Documentation, Release 1.0

6.3. DUIM-Sheets Module

- **number** – An instance of type `<number>`.

**Discussion** Returns the number of pixels per point for `display`.

**See also**
- `display-pixel-height`
- `display-pixel-width`
- `display-units`

**display-pixel-width** Generic function
Returns the width of the specified display in pixels.

**Signature** `display-pixel-width display => width`

**Parameters**
- `display` – An instance of type `<display>`.

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

**Discussion** 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`
- `display-pixel-height`
- `display-units`
- `display-width`

**display-units** Generic function
Returns the default units for the specified display.

**Signature** `display-units display => value`

**Parameters**
- `display` – An instance of type `<display>`.

**Values**
- `value` – An instance of type `one-of(#"device", #"pixels", #"mm")`.

**Discussion** Returns the default units for `display`. 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 `#"default"`, so as to maintain a device-independent measurement as far as possible.

**See also**
- `display-height`
- `display-width`

**display-width** Generic function
Returns the width of the specified display.

**Signature** `display-width display #key units => width`

**Parameters**
• **display** – An instance of type `<display>`.

• **units** – An instance of one-of(#"device", #"mm", #"pixels"). Default value: #"device".

**Values**

• **width** – An instance of type `<number>`.

**Discussion** Returns the width of **display**, in device-independent units. If **units** is specified, then the value returned is converted into the appropriate type of units.

**See also**

• **display-depth**

• **display-height**

• **display-mm-width**

• **display-orientation**

• **display-pixel-width**

• **display-units**

**do-children-containing-position**

Generic function

Invokes a function on any children that occupy a specified position in the specified sheet.

**Signature**

`do-children-containing-position function sheet x y => ()`

**Parameters**

• **function** – An instance of type `<function>`.

• **sheet** – An instance of type `<sheet>`.

• **x** – An instance of type `<real>`.

• **y** – An instance of type `<real>`.

**Discussion** 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**

**do-children-overlapping-region**

Generic function

Invokes a function on any children of the specified sheet whose regions overlap a specified region.

**Signature**

`do-children-overlapping-region function sheet region => ()`

**Parameters**

• **function** – An instance of type `<function>`.

• **sheet** – An instance of type `<sheet>`.

• **region** – An instance of type `<region>`.

**Discussion** Invokes **function** on any children of **sheet** whose regions overlap **region**. This is used by **children-overlapping-region** to ascertain which children overlap **region**.

**See also**

• **children-overlapping-region**
do-displays Function
Runs a function on all the displays attached to a given port.

Signature  do-displays function port => ()
Parameters
• function – An instance of type <function>.
• port – An instance of type <port>.
Discussion Runs a function on all the displays attached to a given port. By default, the current port is used, unless port is specified.

do-frames Generic function
Runs a function on all the frames managed by a given frame manager.

Signature  do-frames function #key port frame-manager => ()
Parameters
• function – An instance of type <function>.
• port – An instance of type <port>.
• frame-manager – An instance of type <frame-manager>.
Discussion 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 port or frame-manager are specified.

do-ports Function
Runs a function on all the current ports.

Signature  do-ports function => ()
Parameters
• function – An instance of type <function>.
Discussion Runs a function on all the current ports.

do-sheet-children Generic function
Runs a function on all the immediate children of the specified sheet.

Signature  do-sheet-children function sheet => ()
Parameters
• function – An instance of type <function>.
• sheet – An instance of type <sheet>.
Discussion Runs function on all the immediate children of sheet. This function calls sheet-children to find the children of sheet.
See also
• sheet-children

do-sheet-tree Generic function
Runs a function on all the children in the hierarchy of the specified sheet.

Signature  do-sheet-tree function sheet => ()
Parameters
• function – An instance of type <function>.
• **sheet** – An instance of type `<sheet>`.

**Discussion** 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> Instantiable Sealed Class**
The class of double-click events on the pointer device.

**Superclasses** `<button-press-event>`

**Discussion** 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 previous 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>`.

**See also**
- `<button-press-event>`

**do-with-drawing-options Generic function**
Runs some code on a drawable in a given drawing context.

**Signature**
do-with-drawing-options **drawable** function **#key** brush pen text-style clipping-region transform => #rest values

**Parameters**
- **drawable** – An instance of type `type-union(<sheet>, <medium>)`.
- **function** – An instance of type `<function>`.
- **brush** – An instance of type `<brush>`.
- **pen** – An instance of type `<pen>`.
- **text-style** – An instance of type `<text-style>`.
- **clipping-region** – An instance of type `<region>`.
- **transform** – An instance of type `<transform>`.

**Values**
- **values** – An instance of type `<object>`.

**Discussion**
Runs some code on a drawable in a given drawing context. This function is called by the macro `with-drawing-options`, and you should define new methods on it for new classes of drawable.

The **function** passed to `do-with-drawing-options` is the result of encapsulating the body passed to `with-drawing-options` as a stand-alone method.

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`

**do-with-pointer-grabbed Generic function**
Runs some specified code, forwarding all pointer events to a sheet.

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

• **port** – An instance of type `<port>`.
• **sheet** – An instance of type `<sheet>`.
• **continuation** – An instance of type `<function>`.

Values

• **values** – An instance of type `<object>`.

Discussion

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**

**do-with-sheet-medium**

**Generic function**

Runs a continuation function on a sheet.

**Signature**

`do-with-sheet-medium sheet continuation => #rest values`

**Parameters**

• **sheet** – An instance of type `<sheet>`.
• **continuation** – An instance of type `<function>`.

**Values**

• **values** – An instance of type `<object>`.

**Discussion**

Runs a continuation function on a sheet.

**See also**

• **with-sheet-medium**

**do-with-text-style**

**Generic function**

Runs some code on a drawable in the context of a given text style.

**Signature**

`do-with-text-style drawable function text-style => ()`

**Parameters**

• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.
• **function** – An instance of type `<function>`.
• **text-style** – An instance of type `<text-style>`.

**Discussion**

Runs some code on a drawable in the context of a given text style.

**See also**

• **with-text-style**

**do-with-transform**

**Generic function**

Returns the result of running a function in a transform defined on a specified medium.

**Signature**

`do-with-transform drawable function transform => #rest values`

**Parameters**
• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.

• **function** – An instance of type `<function>`.

• **transform** – An instance of type `<transform>`.

**Values**

• **values** – An instance of type `<object>`.

**Discussion**

Returns the result of running a function in a transform defined on a specified medium. Methods on this function are called by `with-transform`, which in turn is used by the similar macros `with-rotation`, `with-scaling`, and `with-translation`.

**See also**

• `with-transform`

**<event>** Open Abstract Class

The base class of all DUIM events.

**Superclasses** `<object>`

**Init-Keywords**

• **timestamp** – An instance of type `<integer>`. Default value: `next-event-timestamp()`.

**Discussion**

The base class of all DUIM events.

The `timestamp`: init-keyword is used to give a unique identifier for the event.

**Operations**

• `event?`

• `event-matches-gesture?`

• `handle-event`

• `queue-event`

**See also**

• `<frame-event>`

• `<sheet-event>`

**event?** Generic function

Returns true if the specified object is an event.

**Signature**

```scheme
event? object => event?
```

**Parameters**

• **object** – An instance of type `<object>`.

**Values**

• **event?** – An instance of type `<boolean>`.

**Discussion**

Returns true if `object` is an instance of `<event>` or one of its subclasses.

**See also**

• `<event>`
**event-button** Generic function

Returns an integer corresponding to the mouse button that was pressed or released.

**Signature**

\[ \text{event-button} \ \text{event} \Rightarrow \text{integer} \]

**Parameters**

- \( \text{event} \) – An instance of type \(<\text{event}>\).

**Values**

- \( \text{integer} \) – An instance of type \(<\text{integer}>\).

**Discussion**

Returns an integer corresponding to the mouse button that was pressed or released, which will be one of \(\text{$left-button}^*, \text{$middle-button}^*, \text{$right-button}^*\).  

**Note:** The function \(\text{event-button}^*\) records the button state at the time that the event occurred, and hence can be different from \(\text{pointer-button-state}^*\).

**See also**

- \(\text{$left-button}^*\)
- \(\text{$middle-button}^*\)
- \(<\text{pointer-button-event}^*\>
- \(\text{pointer-button-state}^*\)
- \(\text{$right-button}^*\)

**event-character** Generic function

Returns the character that was pressed on the keyboard.

**Signature**

\[ \text{event-character} \ \text{event} \Rightarrow \text{value} \]

**Parameters**

- \( \text{event} \) – An instance of type \(<\text{event}>\).

**Values**

- \( \text{value} \) – An instance of type \(<\text{false-or(<character>>}\).

**Discussion**

Returns the character associated with the keyboard event, if there is any.

**See also**

- \(\text{event-key-name}^*\)
- \(<\text{keyboard-event}^*\>

**event-key-name** Generic function

Returns the name of the key that was pressed or released on the keyboard.

**Signature**

\[ \text{event-key-name} \ \text{event} \Rightarrow \text{name} \]

**Parameters**

- \( \text{event} \) – An instance of type \(<\text{event}>\).

**Values**

- \( \text{name} \) – An instance of type \(<\text{symbol}>\).
**Discussion**  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**
- `event-character`
- `<keyboard-event>`

**event-matches-gesture?**  Generic function

Returns true if an event matches a defined gesture.

**Signature**  `event-matches-gesture? event gesture-name => matches?`

**Parameters**
- `event` – An instance of type `<event>`.
- `gesture-name` – An instance of type `type-union(<gesture>, <character>)`.

**Values**
- `matches?` – An instance of type `<boolean>`.

**Discussion**  Returns true if an event matches a defined gesture.

**event-modifier-state**  Generic function

Returns an integer value that encodes the state of all the modifier keys on the keyboard.

**Signature**  `event-modifier-state event => integer`

**Parameters**
- `event` – An instance of type `<event>`.

**Values**
- `integer` – An instance of type `<integer>`.

**Discussion**  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 `$shift-key`, `$control-key`, `$meta-key`, `$super-key`, and `$hyper-key`.

**See also**
- `event-sheet`
- `gesture-modifier-state`
- `make-modifier-state`
- `port-modifier-state`

**event-pointer**  Generic function

Returns the pointer object to which the specified event refers.

**Signature**  `event-pointer event => pointer`

**Parameters**
- `event` – An instance of type `<event>`.

**Values**
- `pointer` – An instance of type `<pointer>`.

**Discussion**  Returns the pointer object to which `event` refers.
See also
  • <pointer>
  • event-x
  • event-y

**event-region** Generic function
Returns the region in the sheet that is affected by the specified event.

*Signature*  
`event-region event => region`

*Parameters*
  • `event` – An instance of type `<event>`.

*Values*
  • `region` – An instance of type `<region>`.

*Discussion*  
Returns the region of the sheet that is affected by `event`.

See also
  • `event-x`
  • `event-y`
  • `<window-event>`

**event-sheet** Generic function
Returns the sheet associated with the specified event.

*Signature*  
`event-sheet event => sheet`

*Parameters*
  • `event` – An instance of type `<event>`.

*Values*
  • `sheet` – An instance of type `<sheet>`.

*Discussion*  
Returns the sheet associated with `event`.

See also
  • `event-modifier-state`

**event-x** Generic function
Returns the x position of the pointer at the time the event occurred.

*Signature*  
`event-x event => x`

*Parameters*
  • `event` – An instance of type `<event>`.

*Values*
  • `x` – An instance of type `<integer>`.

*Discussion*  
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
  • `event-pointer`
event-y Generic function

Returns the y position of the pointer at the time the event occurred.

**Signature**

```
event-y event => y
```

**Parameters**

- `event` – An instance of type `<event>`.

**Values**

- `y` – An instance of type `<integer>`.

**Discussion**

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**

- `event-pointer`
- `event-region`
- `event-x`

**find-display Function**

Returns a suitable display for the specified port and server-path criteria.

**Signature**

```
find-display #key server-path port orientation units => display
```

**Parameters**

- `server-path` – An instance of type `<symbol>`. Default value: `#(#"local")`.
- `port` – An instance of type `<port>`.
- `orientation` – An instance of type `one-of(#"default")`. Default value: `#"default"`.
- `units` – An instance of type `one-of(#"device", #"pixels", #"mm")`. Default value: `#"device"`.

**Values**

- `display` – An instance of type `<display>`.

**Discussion**

Returns a suitable display for the specified port and server-path criteria.

The `orientation` and `units` arguments can be used to specify the orientation and display units that the returned `display` needs to use.

**See also**

- `find-port`

**find-frame-manager Function**

Returns a suitable frame manager for the specified criteria.

**Signature**

```
find-frame-manager #rest options #key port server-path class palette => framem
```

**Parameters**

- `options` – An instance of type `<object>`.
• **port** – An instance of type `<port>`.
• **server-path** – An instance of type `<object>`.
• **class** – An instance of type `<type>`.
• **palette** – An instance of type `<palette>`.

**Values**

• **framem** – An instance of type `<frame-manager>`.

**Discussion**

Returns a suitable frame manager for the specified criteria.

If necessary, you can specify a *port*, *server-path*, *class*, or *palette*. If any of these are not specified, then the default value is used in each case. The *class* argument specifies the class of frame manager that should be returned.

**find-port Function**

Returns a suitable port for the specified server-path.

**Signature**

`find-port #rest initargs #key server-path => port`

**Parameters**

• **initargs** – An instance of type `<object>`.
• **server-path** – An instance of type `<object>`. Default value: `*default-server-path*`.

**Values**

• **port** – An instance of type `<port>`.

**Discussion**

Returns a suitable port for the specified server-path.

**See also**

• **find-display**

**fixed-width-font? Generic function**

Returns true if the specified text style uses a fixed-width font.

**Signature**

`fixed-width-font? text-style port #key character-set => fixed?`

**Parameters**

• **text-style** – An instance of type `<text-style>`.
• **port** – An instance of type `<port>`.
• **character-set** – An instance of type `<object>`. Default value: `$standard-character-set`.

**Values**

• **fixed?** – An instance of type `<boolean>`.

**Discussion**

Returns true if *text-style* uses a fixed-width font.

**font-ascent Generic function**

Returns the ascent of the font in the specified text style.

**Signature**

`font-ascent text-style port #key character-set => ascent`

**Parameters**
• **text-style** – An instance of type `<text-style>`.
• **port** – An instance of type `<port>`.
• **character-set** – An instance of type `<object>`. Default value: $standard-character-set$.

Values

• **ascent** – An instance of type `<real>`.

Discussion  Returns the ascent of the font in the `text-style` on `port`.

See also

• **font-descent**
• **font-height**
• **font-metrics**
• **font-width**

**font-descent** Generic function

Returns the descent of the font in the specified text style.

**Signature**  `font-descent text-style port #key character-set => descent`

**Parameters**

• **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>`.

Discussion  Returns the descent of the font in the `text-style` on `port`.

See also

• **font-ascent**
• **font-height**
• **font-metrics**
• **font-width**

**font-height** Generic function

Returns the height of the font in the specified text style.

**Signature**  `font-height text-style port #key character-set => height`

**Parameters**

• **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>`.

Discussion  Returns the height of the font in the `text-style` on `port`.
See also

- `font-ascent`
- `font-descent`
- `font-metrics`
- `font-width`

**font-metrics** Generic function

Returns the metrics of the font in the specified text style.

**Signature**

```
font-metrics text-style port #key character-set => font width height ascent descent
```

**Parameters**

- `text-style` – An instance of type `<text-style>`.
- `port` – An instance of type `<port>`.
- `character-set` – An instance of type `<object>`.

**Values**

- `font` – An instance of type `<object>`.
- `width` – An instance of type `<real>`.
- `height` – An instance of type `<real>`.
- `ascent` – An instance of type `<real>`.
- `descent` – An instance of type `<real>`.

**Discussion**

Returns the metrics of the font in the `text-style` on `port`.

See also

- `font-ascent`
- `font-descent`
- `font-height`
- `font-width`

**font-width** Generic function

Returns the width of the font in the specified text style.

**Signature**

```
font-width text-style port #key character-set => width
```

**Parameters**

- `text-style` – An instance of type `<text-style>`.
- `port` – An instance of type `<port>`.
- `character-set` – An instance of type `<object>`.

**Values**

- `width` – An instance of type `<real>`.

**Discussion**

Returns the width of the font in the `text-style` on `port`.

See also

- `font-ascent`
- `font-descent`
• font-height
• font-metrics

**force-display** Generic function
Forces the specified drawable object to be displayed.

**Signature**  
force-display drawable => ()

**Parameters**

• **drawable** – An instance of type type-union(<sheet>, <medium>).

**Discussion**  
Forces **drawable** to be displayed.

**<frame-event>** Open Abstract Class
The class of events that occur in frames.

**Superclasses** <event>

**Parameter frame** An instance of type <frame>. Required.

**Discussion**  
The class of events that occur in frames. The frame: init-keyword specified the frame in which the event occurs.

**See also**

• <frame-created-event>
• <frame-destroyed-event>
• <frame-exited-event>
• <frame-exit-event>
• <frame-mapped-event>
• <frame-unmapped-event>

**<frame-manager>** Open Abstract Class
The class of frame managers.

**Superclasses** <object>

**Discussion**

The class of frame managers.

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.

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:

1. What concrete gadget to create for an abstract gadget.
2. How to layout the various parts of a frame, such as its menu, tool, and status bars.
3. How to lay out dialogs and their exit buttons.
4. 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*
- *frame-manager?*

**frame-manager** Generic function

Returns the frame manager for the specified object.

**Signature**  
frame-manager object => value

**Parameters**

- **object** – An instance of type <object>.
- **value** – An instance of type false-or(<frame-manager>).

**Discussion**  
Returns the frame manager used to control the look and feel of the display of object.

See also

- *display*
- *<frame-manager>*
- *frame-manager?*
- *port*

**frame-manager?** Generic function

Returns true if the specified object is a frame manager.

**Signature**  
frame-manager? object => framem?

**Parameters**

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

**Values**

- **framem?** – An instance of type <boolean>.

**Discussion**  
Returns true if object is a frame manager.
See also
• <frame-manager>
• frame-manager

frame-manager-frames Generic function
Returns the frames managed by the specified frame manager.

Signature  frame-manager-frames framem => frames

Parameters
• framem – An instance of type <frame-manager>.
• frames – An instance of type limited(<sequence>, of: <frame>).

Discussion  Returns the frames managed by framem.

frame-manager-palette Generic function
Returns the palette used by the specified frame manager.

Signature  frame-manager-palette framem => palette

Parameters
• framem – An instance of type <frame-manager>.

Values
• palette – An instance of type <palette>.

Discussion  Returns the palette used by framem.

See also
• frame-manager-palette-setter

frame-manager-palette-setter Generic function
Sets the palette used by the specified frame manager.

Signature  frame-manager-palette-setter palette framem => palette

Parameters
• palette – An instance of type <palette>.
• framem – An instance of type <frame-manager>.

Values
• palette – An instance of type <palette>.

Discussion  Sets the palette used by framem.

See also
• frame-manager-palette

<gesture> Abstract Instantiable Class
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>`.  

**Discussion**  
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>`
• `<pointer-gesture>`

**gesture-button**  
Generic function  
Returns the button associated with the specified gesture.

**Signature**  
gesture-button pointer-gesture => button

**Parameters**

• **pointer-gesture** – An instance of type `<pointer-gesture>`.

**Values**

• **button** – An instance of type `<integer>`.

**Discussion**  
Returns the button associated with *pointer-gesture*.

**See also**

• `<pointer-gesture>`

**gesture-keysym**  
Generic function  
Returns the keysym associated with the specified gesture.

**Signature**  
gesture-keysym keyboard-gesture => keysym

**Parameters**

• **keyboard-gesture** – An instance of type `<keyboard-gesture>`.

**Values**

• **keysym** – An instance of type `<symbol>`.

**Discussion**  
Returns the keysym associated with *keyboard-gesture*.

**See also**

• `<keyboard-gesture>`

**gesture-modifier-state**  
Generic function  
Returns the modifier-state associated with the specified gesture.

**Signature**  
gesture-modifier-state gesture => modifier-state

**Parameters**
• **gesture** – An instance of type `<gesture>`.

Values

• **modifier-state** – An instance of type `<integer>`.

Discussion  Returns the modifier-state associated with `gesture`.

See also

• `event-modifier-state`
• `<keyboard-gesture>`
• `make-modifier-state`
• `port-modifier-state`

**gesture-spec-equal Function**

Returns true if the two specified gestures are equivalent.

**Signature**  `gesture-spec-equal gesture1 gesture2 => equal?`

**Parameters**

• `gesture1` – An instance of type `<gesture>`.
• `gesture2` – An instance of type `<gesture>`.

Values

• `equal?` – An instance of type `<boolean>`.

Discussion  Returns true if `gesture1` and `gesture2` are equivalent.

See also

• `=`

**get-clipboard-data-as Generic function**

Returns data of a given type from a clipboard.

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

**Parameters**

• `type` – An instance of `type-union(<symbol>, <type>)`.
• `clipboard` – An instance of `<clipboard>`.

Values

• `data` – Instances of `<object>`.

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

See also

• `add-clipboard-data`
• `add-clipboard-data-as`
• `<clipboard>`
• `clipboard-data-available?`

**get-default-background Generic function**

Returns the default background for the specified sheet.
Signature  get-default-background \textit{port sheet} \#key \textit{background} \Rightarrow \textit{background}

Parameters

- \textit{port} – An instance of type \texttt{<port>}.  
- \textit{sheet} – An instance of type \texttt{<sheet>}.  
- \textit{background} – An instance of type \texttt{<ink>}.  

Values

- \textit{background} – An instance of type \texttt{<ink>}.  

Discussion

Returns the default background for \textit{sheet} on \textit{port}.  
If \textit{background} is specified, then this is used instead of the default.  

See also

- \texttt{get-default-foreground}  
- \texttt{get-default-text-style}  

\texttt{get-default-foreground} Generic function  
Returns the default foreground for the specified sheet.

Signature  get-default-foreground \textit{port sheet} \#key \textit{foreground} \Rightarrow \textit{foreground}

Parameters

- \textit{port} – An instance of type \texttt{<port>}.  
- \textit{sheet} – An instance of type \texttt{<sheet>}.  
- \textit{foreground} – An instance of type \texttt{<ink>}.  

Values

- \textit{foreground} – An instance of type \texttt{<ink>}.  

Discussion

Returns the default foreground for \textit{sheet} on \textit{port}.  
If \textit{foreground} is specified, then this is used instead of the default.  

See also

- \texttt{get-default-background}  
- \texttt{get-default-text-style}  

\texttt{get-default-text-style} Generic function  
Returns the default text style for the specified sheet.

Signature  get-default-text-style \textit{port sheet} \#key \textit{text-style} \Rightarrow \textit{text-style}

Parameters

- \textit{port} – An instance of type \texttt{<port>}.  
- \textit{sheet} – An instance of type \texttt{<sheet>}.  
- \textit{text-style} – An instance of type \texttt{<text-style>}.  

Values

- \textit{text-style} – An instance of type \texttt{<text-style>}.  

6.3. DUIM-Sheets Module
Discussion

Returns the default text style for sheet on port.

If text-style is specified, then this is used instead of the default.

See also

- get-default-background
- get-default-foreground

handle-event Generic function

Implements any defined policies of the specified sheet with respect to the specified event.

Signature handle-event sheet event => ()

Parameters

- sheet – An instance of type <sheet>.
- event – An instance of type <event>.

Discussion

Implements any defined policies of sheet with respect to event. Methods defined on this generic are called by DUIM to do the handling.

For example, to highlight a sheet in response to an event that informs the sheet when the pointer has entered the region it occupies, there should be a method to carry out the policy that specializes the appropriate sheet and event classes.

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

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

See also

- handle-repaint
- queue-event

handle-repaint Generic function

Implements region repainting for a given sheet class.

Signature handle-repaint sheet medium region => ()

Parameters

- sheet – An instance of type <sheet>.
- medium – An instance of type <medium>.
- region – An instance of type <region>.

Discussion

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

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

The *sheet* on *medium* is repainted and *region* is the region to repaint.

**See also**
- *<drawing-pane>*
- *pane-display-function*
- *queue-repaint*
- *repaint-sheet*
- *<simple-pane>*
- *<window-repaint-event>*

**$hyper-key** Constant

A constant that represents the HYPER key on the keyboard.

**Type** <integer>

**Value** ash(1, %modifier_base + 4);

**Discussion** A constant that represents the HYPER key on the keyboard.

**See also**
- *$alt-key*
- *$control-key*
- *$meta-key*
- *modifier-key-index*
- *modifier-key-index-name*
- *$modifier-keys*
- *$option-key*
- *$shift-key*
- *$super-key*

**<keyboard-event>** Open Abstract Class

The base class of all keyboard events.

**Superclasses** *<device-event>*

**Init-Keywords**

- **key-name** – An instance of type false-or(<symbol>). Default value: #f.
- **character** – An instance of type false-or(<character>). Default value: #f.
Discussion

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 character: init-keyword represents the keyboard character that was pressed for characters in the standard character set.

Operations

• event-character
• event-key-name
• event-matches-gesture?

See also

• event-character
• event-key-name
• <key-press-event>
• <key-release-event>

<keyboard-gesture> Instantiable Sealed Class

The base class of all keyboard gestures.

Superclasses <gesture>

Init-Keywords

• keysym – An instance of type <symbol>.
• modifier-state – An instance of type <integer>.

Discussion

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
• gesture-modifier-state

<key-press-event> Instantiable Sealed Class

The class of events passed when a key is pressed.

Superclasses <keyboard-event>

Discussion The class of events passed when a key is pressed.

Operations

See also

• <keyboard-event>
• <key-release-event>
<key-release-event> Instantiable Sealed Class
   The class of events passed when a key is released.

   Superclasses <keyboard-event>

   Discussion  The class of events passed when a key is released.

   Operations

   See also
   • <keyboard-event>
   • <key-press-event>

$left-button Constant
   A constant that represents the left button on the attached pointing device.

   Type <integer>

   Value  ash(1, %button_base + 0)

   Discussion  A constant that represents the left button on the attached pointing device.

   See also
   • $middle-button
   • $pointer-buttons
   • $right-button

lower-sheet Generic function
   Lowers the specified sheet to the bottom of the current hierarchy of sheets.

   Signature  lower-sheet sheet => ()

   Parameters
   • sheet – An instance of type <sheet>.

   Discussion  Lowers sheet to the bottom of the current hierarchy of sheets.

   See also
   • lower-frame
   • raise-frame
   • raise-sheet

make-frame-manager Generic function
   Returns an instance of <frame-manager> on the specified port.

   Signature  make-frame-manager port #key palette => framem

   Parameters
   • port – An instance of type <port>.
   • palette – An instance of type <palette>.
   • framem – An instance of type <frame-manager>.

   Discussion  Returns an instance of <frame-manager> on port. If specified, the palette described
   by palette is used.

   See also

6.3. DUIM-Sheets Module
• `<frame-manager>`

**make-modifier-state** Function
Returns a modifier state for the specified modifiers.

**Signature**  
`make-modifier-state #rest modifiers => integer`

**Parameters**
- `modifiers` – An instance of type `limited(sequence, of: integer)`.

**Values**
- `integer` – An instance of type `<integer>`.

**Discussion**  
Returns a modifier state for `modifiers`.

**See also**
- `event-modifier-state`
- `gesture-modifier-state`
- `port-modifier-state`

**make-pane** Generic function
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`

**Parameters**
- `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>`.

**Discussion**  
Selects a class that implements the behavior of `pane-class` and constructs a pane of that class.

**<medium>** Open Abstract Instantiable Class
The class of all mediums.

**Superclasses** `<object>`

**Discussion**
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-background-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
• port
• synchronize-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?
• <pixmap-medium>

medium? Generic function
Returns true if the specified object is a medium.

Signature  medium? object => medium?

Parameters
• object – An instance of type <object>.

Values
• medium? – An instance of type <boolean>.

Discussion  Returns true if object is a medium.

See also
• <medium>
• sheet?

medium-background Generic function
Returns the background for the specified medium.

Signature  medium-background medium => ink

Parameters
• medium – An instance of type <medium>.

Values
• ink – An instance of type <ink>.

Discussion  Returns the background for medium.

See also
• medium-background-setter
• medium-foreground
**medium-background-setter** Generic function
Sets the background for the specified medium.

**Signature** medium-background-setter \( background \ medium \Rightarrow background \)

**Parameters**
- **background** – An instance of type \(<ink>\).
- **medium** – An instance of type \(<medium>\).

**Values**
- **background** – An instance of type \(<ink>\).

**Discussion** Sets the background for \( medium \).

**See also**
- **medium-background**
- **medium-foreground-setter**

**medium-brush** Generic function
Returns the brush for the specified medium.

**Signature** medium-brush \( medium \Rightarrow brush \)

**Parameters**
- **medium** – An instance of type \(<medium>\).

**Values**
- **brush** – An instance of type \(<brush>\).

**Discussion** Returns the brush for \( medium \). This brush is used by all subsequent painting operations on \( medium \).

**See also**
- **medium-brush-setter**
- **medium-pen**

**medium-brush-setter** Generic function
Sets the brush for the specified medium.

**Signature** medium-brush-setter \( brush \ medium \Rightarrow brush \)

**Parameters**
- **brush** – An instance of type \(<brush>\).
- **medium** – An instance of type \(<medium>\).

**Values**
- **brush** – An instance of type \(<brush>\).

**Discussion** Sets the brush for \( medium \). This brush is used by all subsequent painting operations on \( medium \).

**See also**
- **medium-brush**
- **medium-pen-setter**
**medium-clipping-region** Generic function

Returns the clipping region for the specified medium.

**Signature**

medium-clipping-region medium => region

**Parameters**

- medium – An instance of type <medium>.

**Values**

- region – An instance of type <region>.

**Discussion**

Returns the clipping region for medium.

**See also**

- medium-clipping-region-setter

**medium-clipping-region-setter** Generic function

Sets the clipping region for the specified medium.

**Signature**

medium-clipping-region-setter region medium => region

**Parameters**

- region – An instance of type <region>.
- medium – An instance of type <medium>.

**Values**

- region – An instance of type <region>.

**Discussion**

Sets the clipping region for medium.

**See also**

- medium-clipping-region

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

Returns the default text style for the specified medium.

**Signature**

medium-default-text-style medium => text-style

**Parameters**

- medium – An instance of type <medium>.

**Values**

- text-style – An instance of type <text-style>.

**Discussion**

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
- medium-merged-text-style
- medium-text-style

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

Sets the default text style for the specified medium.

**Signature**

medium-default-text-style-setter text-style medium => text-style

**Parameters**
• **text-style** – An instance of type `<text-style>`.
• **medium** – An instance of type `<medium>`.

### Values
• **text-style** – An instance of type `<text-style>`.

### Discussion
Sets 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`
• `medium-text-style-setter`

---

**medium-drawable** Generic function

Returns the drawable for the specified medium.

#### Signature
`medium-drawable medium => drawable`

#### Parameters
• **medium** – An instance of type `<medium>`.

#### Values
• **drawable** – An instance of type `<object>`.

#### Discussion
Returns the drawable for `medium`.

#### See also
• `medium-drawable-setter`

---

**medium-drawable-setter** Generic function

Sets the drawable for the specified medium.

#### Signature
`medium-drawable-setter drawable medium => object`

#### Parameters
• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.
• **medium** – An instance of type `<medium>`.

#### Values
• **object** – An instance of type `<object>`.

#### Discussion
Sets the drawable for `medium`.

#### See also
• `medium-drawable`

---

**medium-foreground** Generic function

Returns the foreground of the specified medium.

#### Signature
`medium-foreground medium => ink`

#### Parameters
• **medium** – An instance of type `<medium>`.

#### Values
• **ink** – An instance of type `<ink>`.
**Discussion** Returns the foreground of medium.

**See also**
- medium-background
- medium-foreground-setter

### medium-foreground-setter

**Generic function**
Sets the foreground of the specified medium.

**Signature** medium-foreground-setter foreground medium => foreground

**Parameters**
- **foreground** – An instance of type <ink>.
- **medium** – An instance of type <medium>.

**Values**
- **foreground** – An instance of type <ink>.

**Discussion** Sets the foreground of medium.

**See also**
- medium-background-setter
- medium-foreground

### medium-merged-text-style

**Generic function**
Returns the merged text style of the specified medium.

**Signature** medium-merged-text-style medium => text-style

**Parameters**
- **medium** – An instance of type <medium>.

**Values**
- **text-style** – An instance of type <text-style>.

**Discussion** Returns the merged text style of medium.

**See also**
- medium-default-text-style
- medium-text-style

### medium-pen

**Generic function**
Returns the pen for the specified medium.

**Signature** medium-pen medium => pen

**Parameters**
- **medium** – An instance of type <medium>.

**Values**
- **pen** – An instance of type <pen>.

**Discussion** Returns the pen for medium. This brush is used by all subsequent drawing operations on medium.

**See also**
• medium-brush

• medium-pen-setter

**medium-pen-setter** Generic function
Sets the pen for the specified medium.

**Signature** medium-pen-setter pen medium => pen

**Parameters**

• **pen** – An instance of type `<pen>`.

• **medium** – An instance of type `<medium>`.

**Values**

• **pen** – An instance of type `<pen>`.

**Discussion** Sets the pen for `medium`. This brush is used by all subsequent drawing operations on `medium`.

**See also**

• medium-brush-setter

• medium-pen

**medium-pixmap** Generic function
Returns the pixmap for the specified medium.

**Signature** medium-pixmap medium => value

**Parameters**

• **medium** – An instance of type `<medium>`.

**Values**

• **value** – An instance of type `false-or(<pixmap>)`.

**Discussion** Returns the pixmap for `medium`. This pixmap is used by all subsequent pixmap operations on `medium`.

**See also**

• medium-pixmap-setter

**medium-pixmap-setter** Generic function
Sets the pixmap for the specified medium.

**Signature** medium-pixmap-setter pixmap medium => value

**Parameters**

• **pixmap** – An instance of type `<pixmap>`.

• **medium** – An instance of type `<medium>`.

**Values**

• **value** – An instance of type `false-or(<pixmap>)`.

**Discussion** Returns the pixmap for `medium`. This pixmap is used by all subsequent pixmap operations on `medium`.

**See also**

• medium-pixmap
medium-sheet Generic function
Returns the sheet for the specified medium.

Signature  medium-sheet medium => sheet
Parameters
  • medium – An instance of type <medium>.
Values
  • sheet – An instance of type false-or(<sheet>).
Discussion  Returns the sheet for medium, if there is one.

medium-text-style Generic function
Returns the text style for the specified medium.

Signature  medium-text-style medium => text-style
Parameters
  • medium – An instance of type <medium>.
Values
  • text-style – An instance of type <text-style>.
Discussion  Returns the text style for medium.
See also
  • medium-default-text-style
  • medium-merged-text-style
  • medium-text-style-setter

medium-text-style-setter Generic function
Sets the text style for the specified medium.

Signature  medium-text-style-setter text-style medium => text-style
Parameters
  • text-style – An instance of type <text-style>.
  • medium – An instance of type <medium>.
Values
  • text-style – An instance of type <text-style>.
Discussion  Sets the text style for medium.
See also
  • medium-default-text-style-setter
  • medium-text-style

medium-transform Generic function
Returns the transform for the specified medium.

Signature  medium-transform medium => transform
Parameters
  • medium – An instance of type <medium>. 
Values

• **transform** – An instance of type `<transform>`.

**Discussion** Returns the transform for `medium`.

**See also**

• medium-transform-setter
• sheet-transform

**medium-transform-setter Generic function**

Sets the transform for the specified medium.

**Signature** medium-transform-setter transform medium => transform

**Parameters**

• **transform** – An instance of type `<transform>`.
• **medium** – An instance of type `<medium>`.

**Values**

• **transform** – An instance of type `<transform>`.

**Discussion** Sets the transform for `medium`.

**See also**

• medium-transform
• sheet-transform-setter

**$meta-key Constant**

A constant that represents the META key on the keyboard.

**Type** <integer>

**Value** `ash(1, %modifier_base + 2);`

**Discussion** 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 `$alt-key` is bound to this constant.

**See also**

• $alt-key
• $control-key
• $hyper-key
• modifier-key-index
• modifier-key-index-name
• $modifier-keys
• $option-key
• $shift-key
• $super-key

**$middle-button Constant**

A constant that represents the middle button on the attached pointing device.

**Type** <integer>
Value  \texttt{ash(1, \%button\_base + 1)}

Discussion  A constant that represents the middle button on the attached pointing device.

See also

- \$left\_button
- \$pointer\_buttons
- \$right\_button

\texttt{modifier-key-index} Function

Returns the index number of the specified modifier key.

Signature  \texttt{modifier-key-index key-name => index}

Parameters

- \texttt{key-name} – An instance of type \texttt{symbol}.

Values

- \texttt{index} – An instance of type \texttt{integer}.

Discussion

Returns the index number of the specified modifier key. The \texttt{key-name} specified may be any of the elements of \texttt{modifier-keys}.

The returned index value is either 0, 1, 2, 3, or 4.

See also

- \$alt\_key
- \$control\_key
- \$hyper\_key
- \$meta\_key
- \texttt{modifier-key-index-name}
- \texttt{modifier-keys}
- \$option\_key
- \$shift\_key
- \$super\_key

\texttt{modifier-key-index-name} Function

Returns the key name of the specified modifier key index.

Signature  \texttt{modifier-key-index-name index => key-name}

Parameters

- \texttt{index} – An instance of type \texttt{integer}.

Values

- \texttt{key-name} – An instance of type \texttt{symbol}.

Discussion

Returns the key name of the specified modifier key index. The \texttt{index} specified is either 0, 1, 2, 3, or 4.
The key-name returned may be any of the elements of $modifier-keys

See also
- $alt-key
- $control-key
- $hyper-key
- $meta-key
- modifier-key-index
- $modifier-keys
- $option-key
- $shift-key
- $super-key

$modifier-keys Constant
The default list of keys on the keyboard that are used as modifiers.

Type <sequence>
Value #["shift", "control", "meta", "super", "hyper"]

Discussion The default list of keys on the keyboard that are used as modifiers for keyboard accelerators and mnemonics.

See also
- $alt-key
- $control-key
- $hyper-key
- $meta-key
- modifier-key-index
- modifier-key-index-name
- $option-key
- $shift-key
- $super-key

notify-user Generic function
Creates and displays an alert dialog box with the specified criteria.

Signature notify-user message-string #key frame owner title documentation exit-boxes name style foreground background text-style => boolean

Parameters
- message-string – An instance of type <string>.
- frame – An instance of type <frame>. Default value: current-frame ()
- owner – An instance of type <sheet>.
- title – An instance of type <string>.
- documentation – An instance of type false-or(<string>). Default value: #f.
• **exit-boxes** – An instance of type `<object>`.
• **name** – An instance of type `<object>`.
• **style** – An instance of type `one-of(#"information", #"question", #"warning", #"error", #"serious-error", #"fatal-error")`.
• **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 `false-or(<text-style>)`. Default value: #f.

**Values**

• **boolean** – An instance of type `<boolean>`.

**Discussion**

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.

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*
• *choose-directory*
• *choose-file*

**open-clipboard Function**

Creates a clipboard lock for a sheet on a port.

**Signature**

`open-clipboard port sheet => clipboard`

**Parameters**

• **port** – An instance of `<port>`. 
• **sheet** – An instance of `<sheet>`.

**Values**

• **clipboard** – An instance of `<clipboard>`.

**Discussion**

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

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

**See also**

• `<clipboard>`
• `with-clipboard`

$`option-key` Constant

A constant that represents the OPTION key on the keyboard.

**Type** `<integer>`

**Value** `$super-key`

**Discussion** 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 keyboard.

**See also**

• `$alt-key`
• `$control-key`
• `$hyper-key`
• `$meta-key`
• `modifier-key-index`
• `modifier-key-index-name`
• `$modifier-keys`
• `$shift-key`
• `$super-key`

`<pointer>` Open Abstract Instantiable Class

The class of all pointers.

**Superclasses** `<object>`

**Init-Keywords**

• **port** – An instance of type `<port>`.

**Discussion** The class of all pointers.

**Operations**

The following operations are exported from the `DUIM-Sheets` module.

• `display`
• pointer?
• pointer-button-state
• pointer-cursor
• pointer-cursor-setter
• pointer-position
• pointer-sheet
• port
• set-pointer-position

See also
• pointer?

pointer? Generic function
Returns true if the specified object is a pointer.

Signature  pointer? object => pointer?

Parameters
• object – An instance of type <object>.

Values
• pointer? – An instance of type <boolean>.

Discussion  Returns true if object is a pointer.

See also
• <pointer>

<pointer-boundary-event> Instantiable Sealed Class
The class that corresponds to a pointer motion event that crosses a sheet boundary.

Superclasses  <pointer-motion-event>

Init-Keywords


Discussion

The class that corresponds to a pointer motion event that crosses some sort of sheet boundary.

The kind: 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 DUIM-Sheets module.

• boundary-event-kind

See also
• boundary-event-kind
• <pointer-enter-event>
• <pointer-exit-event>
Open Abstract Class

The class of events that occur when mouse buttons are pressed.

Superclasses <pointer-event>

Init-Key Words

• button – An instance of type one-of($left-button, $middle-button, $right-button).

Discussion The class of events that occur when mouse buttons are pressed.

Operations

The following operations are exported from the DUIM-Sheets module.

• event-button
• event-matches-gesture?
• handle-event

See also

• event-button
• $left-button
• $middle-button
• pointer-button-state
• <pointer-drag-event>
• $right-button

$pointer-buttons Constant

The constant representing the possible buttons on the pointing device.

Type <sequence>

Value #["left", "middle", "right”];

Discussion

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
• button-index-name
• $left-button
• $middle-button
• $right-button

pointer-button-state Generic function

Returns the state of the specified pointer.

Signature pointer-button-state pointer => integer

Parameters
• **pointer** – An instance of type `<pointer>`.

**Values**

• **integer** – An instance of type `<integer>`.

**Discussion** Returns the state of **pointer**.

**pointer-cursor** Generic function

Returns the cursor used for the specified pointer.

**Signature**  
`pointer-cursor pointer => cursor`

**Parameters**

• **pointer** – An instance of type `<pointer>`.

**Values**

• **cursor** – An instance of type `<cursor>`.

**Discussion** Returns the cursor used for **pointer**.

**See also**

• **pointer-cursor-setter**

**pointer-cursor-setter** Generic function

Sets the cursor used for the specified pointer.

**Signature**  
`pointer-cursor-setter cursor pointer => cursor`

**Parameters**

• **cursor** – An instance of type `<cursor>`.

• **pointer** – An instance of type `<pointer>`.

**Values**

• **cursor** – An instance of type `<cursor>`.

**Discussion** Sets the cursor used for **pointer**.

**See also**

• **pointer-cursor**

**<pointer-drag-event>** Instantiable Sealed Class

The class of events describing drag movements.

**Superclasses**  
`<pointer-motion-event> <pointer-button-event>`

**Init-Keywords**

• **button** – An instance of type `one-of($left-button, $middle-button, $right-button)`.

**Discussion**

The class of events describing drag movements. This is the same as `<pointer-motion-event>`, except that a button on the attached pointing device must also be held down as the pointer is moving.

The **button**: init-keyword is inherited from the superclass `<pointer-button-event>`.

**Operations**

**See also**
<pointer-enter-event> Instantiable Sealed Class
The class of events that describe a pointer entering an area such as a sheet.

**Superclasses** <pointer-boundary-event>

**Discussion** The class of events that describe a pointer entering an area such as a sheet.

**Operations**

**See also**

• <pointer-exit-event>

<pointer-event> Open Abstract Class
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>.

**Discussion**

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**

**See also**

• <pointer-button-event>
• <pointer-exit-event>
• <pointer-motion-event>

<pointer-exit-event> Instantiable Sealed Class
The class of events that describe a pointer leaving an area such as a sheet.

**Superclasses** <pointer-boundary-event>

**Discussion** The class of events that describe a pointer leaving an area such as a sheet.

**Operations**

**See also**

• <pointer-button-event>
• <pointer-enter-event>
• <pointer-motion-event>

<pointer-gesture> Instantiable Sealed Class
The class of all gestures that occur on pointers.

**Superclasses** <gesture>

**Init-Keywords**

• button – An instance of type <integer>.
• **modifier-state** – An instance of type `<integer>`.

**Discussion**

The class of all gestures that occur on pointers.

The **button**: init-keyword specifies the button on the attached pointer device on which the gesture has occurred, and the **modifier-state**: init-keyword specifies the modifier-state of the gesture.

**Operations**

• **gesture-button**

<pointer-motion-event> Instantiable Sealed Class

The class of events that describe a pointer that is moving.

**Superclasses** `<pointer-event>`

**Discussion** The class of events that describe a pointer that is moving.

**Operations**

See also

• `<pointer-button-event>`
  • `<pointer-drag-event>`
  • `<pointer-enter-event>`
  • `<pointer-event>`
  • `<pointer-exit-event>`

**pointer-position** Generic function

Returns the current position of the specified pointer.

**Signature** `pointer-position pointer #key sheet => x y`

**Parameters**

• **pointer** – An instance of type `<pointer>`.

• **sheet** – An instance of type `<sheet>`.

**Values**

• **x** – An instance of type `<real>`.

• **y** – An instance of type `<real>`.

**Discussion** Returns the current position of **pointer**. If **sheet** is specified, then the pointer must be over it.

See also

• **pointer-sheet**
  • **set-pointer-position**

**pointer-sheet** Generic function

Returns the sheet under the specified pointer.

**Signature** `pointer-sheet pointer => sheet`

**Parameters**

• **pointer** – An instance of type `<pointer>`.
• **sheet** – An instance of type `false-or(<sheet>)`.

**Discussion** Returns the sheet under `pointer`, or `#f` if there is no sheet under the pointer.

**See also**

• `pointer-position`

**<port> Open Abstract Class**

The class of all ports.

**Superclasses** `<object>`

**Discussion**

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:

• A primary input device (usually a keyboard)
• A pointing device, such as a mouse or trackball
• An event processor that dispatched events to the appropriate sheet.

**Operations**

The following operations are exported from the `DUIM-Sheets` 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>`
• `<sheet>`
**port** Generic function

Returns the port for the specified object.

**Signature**  
port object => value

**Parameters**

- **object** – An instance of type `<object>`.
- **value** – An instance of type `false-or(<port>)`.

**Discussion**  
Returns the port used to display `object`.

**See also**

- `display`
- `frame-manager`
- `<port>`
- `port?`

**port?** Generic function

Returns true if the specified object is a port.

**Signature**  
port? object => boolean

**Parameters**

- **object** – An instance of type `<object>`.

**Values**

- **boolean** – An instance of type `<boolean>`.

**Discussion**  
Returns true if `object` is a port.

**See also**

- `<port>`
- `<port>`

**port-modifier-state** Generic function

Returns the modifier state of the specified port.

**Signature**  
port-modifier-state port => integer

**Parameters**

- **port** – An instance of type `<port>`.

**Values**

- **integer** – An instance of type `<integer>`.

**Discussion**  
Returns the modifier state of `port`.

**See also**

- `event-modifier-state`
- `gesture-modifier-state`
- `make-modifier-state`
- `port-name`
- `port-pointer`
port-name Generic function
Returns the name of the specified port.

Signature  port-name port => name
Parameters
• port – An instance of type <port>.
Values
• name – An instance of type <object>.
Discussion  Returns the name of port.
See also
• port-modifier-state
• port-pointer
• port-server-path
• port-type

port-pointer Generic function
Returns the pointer used on the specified port.

Signature  port-pointer port => pointer
Parameters
• port – An instance of type <port>.
Values
• pointer – An instance of type <pointer>.
Discussion  Returns the pointer used on port.
See also
• port-modifier-state
• port-name
• port-server-path
• port-type

port-server-path Generic function
Returns the server path of the specified port.

Signature  port-server-path port => object
Parameters
• port – An instance of type <port>.
Values
• object – An instance of type <object>.
Discussion  Returns the server path of port.
See also
• port-modifier-state
• port-name
• port-pointer
• port-type

<port-terminated-event> Instantiable Sealed Class
The class of events that describe the termination of a port.

Superclasses  <frame-event>

Init-Keywords
• condition – An instance of type <condition>. Required.

Discussion
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

port-type Generic function
Returns the type of the specified port.

Signature  port-type port => type

Parameters
• port – An instance of type <port>.

Values
• type – An instance of type <symbol>.

Discussion  Returns the type of port.

See also
• port-modifier-state
• port-name
• port-pointer
• port-server-path

queue-event Generic function
Queues an event for the specified sheet.

Signature  queue-event sheet event => ()

Parameters
• sheet – An instance of type <sheet>.
• event – An instance of type <event>.

Discussion  Queues event on the event-queue for sheet.

See also
• handle-event

queue-repaint Generic function
Queues a repaint for the specified region of the specified sheet.
**Signature**  queue-repaint  \textit{sheet region} => ()

**Parameters**

- **\textit{sheet}** – An instance of type \textit{<sheet>}.  
- **\textit{region}** – An instance of type \textit{<region>}.  

**Discussion**  Queues a repaint for the area** of \textit{sheet} defined by \textit{region}.  

See also

- handle-repaint  
- repaint-sheet  
- \textit{<window-repaint-event>}

**raise-sheet** Generic function  

Raises the specified sheet to the top of the current hierarchy of sheets.  

**Signature**  raise-sheet  \textit{sheet} => ()

**Parameters**

- **\textit{sheet}** – An instance of type \textit{<sheet>}.  

**Discussion**  Raises \textit{sheet} to the top of the current hierarchy of sheets.  

See also  

- lower-frame  
- lower-sheet  
- raise-frame

**remove-child** Generic function  

Removes a child from the specified sheet.  

**Signature**  remove-child  \textit{sheet child} => \textit{sheet}

**Parameters**

- **\textit{sheet}** – An instance of type \textit{<sheet>}.  
- **\textit{child}** – An instance of type \textit{<sheet>}.  

**Values**

- **\textit{sheet}** – An instance of type \textit{<sheet>}.  

**Discussion**  Removes \textit{child} from \textit{sheet}. The remaining children in the sheet are laid out again appropriately.  

See also

- add-child  
- replace-child

**repaint-sheet** Generic function  

Repaints the specified region of a sheet.  

**Signature**  repaint-sheet  \textit{sheet region} #key \textit{medium} => ()

**Parameters**

- **\textit{sheet}** – An instance of type \textit{<sheet>}.  

• **region** – An instance of type `<region>`.
• **medium** – An instance of type `<medium>`.

**Discussion** Repaints the area of `sheet` defined by `region`. If specified, the appropriate `medium` is used.

**See also**
• `handle-repaint`
• `queue-repaint`
• `<window-repaint-event>`

**replace-child** Generic function
Replaces a child from the specified sheet with a new one.

**Signature** replace-child `sheet old-child new-child => sheet`

**Parameters**
• **sheet** – An instance of type `<sheet>`.
• **old-child** – An instance of type `<object>`.
• **new-child** – An instance of type `<object>`.

**Values**
• **sheet** – An instance of type `<sheet>`.

**Discussion** Replaces `old-child` with `new-child` in `sheet`. The children in the sheet are laid out again appropriately.

**See also**
• `add-child`
• `remove-child`

**$right-button** Constant
A constant that represents the right button on the attached pointing device.

**Type** `<integer>`

**Value** `ash(1, %button_base + 2)`

**Discussion** A constant that represents the right button on the attached pointing device.

**See also**
• `$left-button`
• `$middle-button`
• `$pointer-buttons`

**set-caret-position** Generic function
Sets the position of the specified cursor.

**Signature** set-cursor-position `cursor x y => ()`

**Parameters**
• **cursor** – An instance of type `<caret>`.
• **x** – An instance of type `<real>`.
• **y** – An instance of type `<real>`.
Discussion  Sets the position of cursor to \((x, y)\).

See also
- caret-position
- set-pointer-position

**set-pointer-position** Generic function
Sets the position of the specified pointer.

**Signature**  
```lisp
set-pointer-position pointer x y #key sheet => ()
```

**Parameters**
- **pointer** – An instance of type `<pointer>`.
- **x** – An instance of type `<real>`.
- **y** – An instance of type `<real>`.
- **sheet** – An instance of type `<sheet>`.

Discussion  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
- set-pointer-position

**set-sheet-edges** Generic function
Sets the edges of the specified sheet relative to its parent.

**Signature**  
```lisp
set-sheet-edges sheet left top right bottom => ()
```

**Parameters**
- **sheet** – An instance of type `<sheet>`.
- **left** – An instance of type `<integer>`.
- **top** – An instance of type `<integer>`.
- **right** – An instance of type `<integer>`.
- **bottom** – An instance of type `<integer>`.

Discussion  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
- set-sheet-size
- sheet-edges

**set-sheet-position** Generic function
Sets the position of the specified sheet relative to its parent.

**Signature**  
```lisp
set-sheet-position sheet x y => ()
```

**Parameters**
- **sheet** – An instance of type `<sheet>`.
- **x** – An instance of type `<real>`.
• y – An instance of type <real>.

Discussion Sets the position of sheet to (x, y) relative to the position of its parent. The layout of sheet is recalculated automatically.

See also
• set-sheet-edges
• set-sheet-size
• sheet-position

set-sheet-size Generic function
Sets the size of the specified sheet.

Signature set-sheet-size sheet width height => ()

Parameters
• sheet – An instance of type <sheet>.
• width – An instance of type <integer>.
• height – An instance of type <integer>.

Discussion Sets the size of sheet. The layout of sheet is recalculated automatically.

See also
• set-sheet-edges
• set-sheet-position

<sheet> Open Abstract Class
The base object class for DUIM windows.

Superclasses <object>

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(style-descriptor). 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>).
• parent – An instance of type false-or(<sheet>). Default value: #f.
• child – An instance of type false-or(<sheet>). Default value: #f.
• children – An instance of type limited(<sequence>, of: <sheet>). Default value: [[]].
• x – An instance of type <integer>.
• y – An instance of type <integer>.
• withdrawn? – An instance of type <boolean>. Default value: #f.
• accepts-focus? – An instance of type `<boolean>`. Default value: #t.

• cursor – An instance of type `<cursor>`.

• caret – An instance of type `type-union(<caret>, one-of(#f, #t))`. Default value: #f.

• foreground – An instance of type `<ink>`.

• background – An instance of type `<ink>`.

• text-style – An instance of type `<text-style>`.

• fixed-width? – An instance of type `<boolean>`.

• fixed-height? – An instance of type `<boolean>`.

• resizable? – An instance of type `<boolean>`.

Discussion

The `port:` init-keyword is true if the pane (and its mirror, if it has one) has been mapped, #f otherwise. In this case, the term mapped means visible on the display, ignoring issues of occlusion.

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.

The `parent:`, `child:`, and `children:` 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.

The `x:` and `y:` init-keywords specify the initial position of the sheet relative to its parent. When `accepts-focus?:` is true, the sheet will accept the pointer focus.

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`

6.3. DUIM-Sheets Module
- child-containing-position
- children-overlapping-region
- clear-box
- 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-foreground
- 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-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>
• <port>

**sheet?** Generic function
Returns true if the specified object is a sheet.

**Signature**  
`sheet? object => boolean`

**Parameters**

• `object` – An instance of type `<object>`.

**Values**

• `boolean` – An instance of type `<boolean>`.

**Discussion**  Returns true if `object` is a sheet.

See also

• `medium?`

**sheet-ancestor?** Generic function
Returns true if the specified sheet has the specified ancestor.

**Signature**  
`sheet-ancestor? sheet putative-ancestor => boolean`

**Parameters**

• `sheet` – An instance of type `<sheet>`.
• `putative-ancestor` – An instance of type `<sheet>`.
Values

- boolean – An instance of type `<boolean>`.

Discussion Returns true if `putative-ancestor` is an ancestor of `sheet`.

See also

- `sheet?`

`sheet-child` Generic function

Returns the child of the specified sheet.

Signature `sheet-child sheet => child`

Parameters

- `sheet` – An instance of type `<sheet>`.
- `child` – An instance of type `false-or(<sheet>)`.

Discussion Returns the child of `sheet`.

See also

- `sheet-children`
- `sheet-child-setter`

`sheet-children` Generic function

Returns a list of sheets that are the children of the specified sheet.

Signature `sheet-children sheet => sheets`

Parameters

- `sheet` – An instance of type `<sheet>`.
- `sheets` – An instance of type `limited(<sequence>, of: <sheet>)`.

Discussion Returns a list of sheets that are the children of `sheet`. Some sheet classes support only a single child; in this case, the return value of `sheet-children` is a list of one element.

See also

- `do-sheet-children`
- `sheet-child`
- `sheet-children-setter`

`sheet-children-setter` Generic function

Sets the children of the specified sheet.

Signature `sheet-children-setter children sheet => sheets`

Parameters

- `children` – An instance of type `limited(<sequence>, of: <sheet>)`.
- `sheet` – An instance of type `<sheet>`.
- `children` – An instance of type `limited(<sequence>, of: <sheet>)`.

Discussion Sets the children of `sheet`. Some sheet classes support only a single child; in this case, `children` is a list of one element.

See also

- `sheet-children`
• sheet-child-setter

**sheet-child-setter** Generic function
Sets the child of the specified sheet.

**Signature**  
(sheet-child-setter)  
child sheet => child

**Parameters**

- **child** – An instance of type <sheet>.
- **sheet** – An instance of type <sheet>.
- **child** – An instance of type false-or(<sheet>).

**Discussion**  
Sets the child of sheet.

See also
- **sheet-child**
- **sheet-children-setter**

**sheet-edges** Generic function
Returns the edges of the specified sheet, relative to its parent.

**Signature**  
(sheet-edges)  
sheet => left top right bottom

**Parameters**

- **sheet** – An instance of type <sheet>.
- **left** – An instance of type <coordinate>.
- **top** – An instance of type <coordinate>.
- **right** – An instance of type <coordinate>.
- **bottom** – An instance of type <coordinate>.

**Discussion**  
Returns the edges of sheet. Each edge is specified relative to the corresponding edge of the parent of sheet.

See also
- **set-sheet-edges**
- **sheet-position**
- **sheet-size**
- **sheet-transform**

**<sheet-event>** Open Abstract Class
The class of events that can occur in sheets.

**Superclasses** <event>

**Init-Keywords**

- **sheet** – An instance of type false-or(<sheet>). Required.

**Discussion**

The class of events that can occur in sheets.

The required init-keyword sheet: specifies a sheet in which the event occurs.
Operations

The following operation is exported from the DUIM-Sheets module.

- `event-sheet`

See also

- `<device-event>`

sheet-event-mask Generic function

Returns the event mask of the specified sheet.

Signature  sheet-event-mask `sheet` => `integer`

Parameters

- `sheet` – An instance of type `<sheet>`.

Values

- `integer` – An instance of type `<integer>`.

Discussion  Returns the event mask of `sheet`.

See also

- `sheet-event-mask-setter`
- `sheet-event-queue`

sheet-event-mask-setter Generic function

Sets the event mask of the specified sheet.

Signature  sheet-event-mask-setter `mask` `sheet` => `mask`

Parameters

- `mask` – An instance of type `<integer>`.
- `sheet` – An instance of type `<sheet>`.

Values

- `mask` – An instance of type `<integer>`.

Discussion  Sets the event mask of `sheet`.

See also

- `sheet-event-mask`

sheet-event-queue Generic function

Returns the event queue of the specified sheet.

Signature  sheet-event-queue `sheet` => `event-queue`

Parameters

- `sheet` – An instance of type `<sheet>`.

Values

- `event-queue` – An instance of type `<event-queue>`.

Discussion  Returns the event mask of `sheet`. This is a list of all the events that are currently queued ready for execution.

See also
• sheet-event-mask

**sheet-frame** **Generic function**

Returns the frame associated with the specified sheet.

**Signature**  

(sheet-frame sheet => frame)

**Parameters**

- sheet – An instance of type <sheet>.
- frame – An instance of type false-or(<frame>).

**Discussion**  

Returns the frame associated with sheet.

See also

- sheet-medium
- sheet-parent

**sheet-mapped?** **Generic function**

Returns true if the specified sheet is mapped.

**Signature**  

(sheet-mapped? sheet => mapped?)

**Parameters**

- sheet – An instance of type <sheet>.

**Values**

- mapped? – An instance of type <boolean>.

**Discussion**  

Returns true if sheet is mapped, that is, displayed on screen (issues of occluding windows notwithstanding).

See also

- sheet-mapped?-setter
- sheet-withdrawn?

**sheet-mapped?-setter** **Generic function**

Specifies whether the specified sheet is mapped.

**Signature**  

(sheet-mapped?-setter mapped? sheet => boolean)

**Parameters**

- mapped? – An instance of type <boolean>.
- sheet – An instance of type <sheet>.

**Values**

- boolean – An instance of type <boolean>.

**Discussion**  

Specifies whether sheet is mapped, that is, displayed on screen (issues of occluding windows notwithstanding). If #t, sheet is mapped, if #f, it is not.

See also

- sheet-mapped?

**sheet-medium** **Generic function**

Returns the medium associated with the specified sheet.

**Signature**  

(sheet-medium sheet => medium)
Parameters

- **sheet** – An instance of type `<sheet>`.
- **medium** – An instance of type `false-or(<medium>)`.

Discussion  Returns the medium associated with `sheet`.

See also

- `sheet-frame`

**sheet-parent**  Generic function

Returns the parent of the specified sheet.

**Signature**  
`sheet-parent sheet => parent`

**Parameters**

- **sheet** – An instance of type `<sheet>`.
- **parent** – An instance of type `false-or(<sheet>)`.

Discussion  Returns the parent of `sheet`.

See also

- `sheet-medium`
- `sheet-parent-setter`
- `sheet-position`

**sheet-parent-setter**  Generic function

Sets the parent of the specified sheet.

**Signature**  
`sheet-parent-setter parent sheet => value`

**Parameters**

- **parent** – An instance of type `false-or(<sheet>)`.
- **sheet** – An instance of type `<sheet>`.

**Values**

- **value** – An instance of type `false-or(<sheet>)`.

Discussion  Sets the parent of `sheet`.

See also

- `sheet-parent`

**sheet-pointer-cursor**  Generic function

Returns the pointer cursor associated with the specified sheet.

**Signature**  
`sheet-pointer-cursor sheet => cursor`

**Parameters**

- **sheet** – An instance of type `<sheet>`.

**Values**

- **cursor** – An instance of type `<cursor>`.

Discussion  Returns 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-setter`
- `sheet-text-cursor`

**sheet-pointer-cursor-setter** Generic function
Sets the pointer cursor associated with the specified sheet.

**Signature**
\[ \text{sheet-pointer-cursor-setter cursor sheet} \Rightarrow \text{cursor} \]

**Parameters**
- `cursor` – An instance of type `<cursor>`.
- `sheet` – An instance of type `<sheet>`.

**Values**
- `cursor` – An instance of type `<cursor>`.

**Discussion**
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`

**sheet-position** Generic function
Returns the position of the specified sheet relative to its parent.

**Signature**
\[ \text{sheet-position sheet} \Rightarrow x \ y \]

**Parameters**
- `sheet` – An instance of type `<sheet>`.

**Values**
- `x` – An instance of type `<real>`.
- `y` – An instance of type `<real>`.

**Discussion**
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`
- `sheet-edges`
- `sheet-parent`
- `sheet-size`
- `sheet-transform`

**sheet-region** Generic function
Returns the region associated with the specified sheet.

**Signature**
\[ \text{sheet-region sheet} \Rightarrow \text{region} \]

**Parameters**
- `sheet` – An instance of type `<sheet>`.

**Values**
- `sheet` – An instance of type `<sheet>`.
• region – An instance of type <region>.

Discussion Returns an instance of <region> that represents the set of points to which sheet refers. The region is expressed in the same coordinate system as sheet.

See also
• sheet-region-setter

sheet-region-setter Generic function
Sets the region associated with the specified sheet.

Signature sheet-region-setter region sheet => region

Parameters
• region – An instance of type <region>.
• sheet – An instance of type <sheet>.

Values
• region – An instance of type <region>.

Discussion Creates or modifies an instance of <region> that represents the set of points to which sheet refers. The region is expressed in the same coordinate system as sheet.

See also
• sheet-region

sheet-size Generic function
Returns the width and height of the specified sheet.

Signature sheet-size sheet => width height

Parameters
• sheet – An instance of type <sheet>.

Values
• width – An instance of type <integer>.
• height – An instance of type <integer>.

Discussion Returns the width and height of the specified sheet. Use set-sheet-size to set or modify the size of a sheet.

See also
• set-sheet-size
• sheet-edges
• sheet-position
• sheet-transform

sheet-state Generic function
Returns the current state of the specified sheet.

Signature sheet-state sheet => value

Parameters
• sheet – An instance of type <sheet>.

Values
• **value** – An instance of type one-of("withdrawn", "managed", "mapped", "unknown").

**Discussion** Returns the current state of `sheet`. 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

Returns the text cursor associated with the specified sheet.

**Signature** `sheet-text-cursor sheet => text-cursor`

**Parameters**

• `sheet` – An instance of type `<sheet>`.

• `text-cursor` – An instance of type `false-or(<cursor>)`.

**Discussion** Returns the text cursor associated with `sheet`. 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**

• `sheet-pointer-cursor`

**sheet-transform** Generic function

Returns the transform associated with the specified sheet.

**Signature** `sheet-transform sheet => transform`

**Parameters**

• `sheet` – An instance of type `<sheet>`.

**Values**

• `transform` – An instance of type `<transform>`.

**Discussion** Returns the transform associated with `sheet`.

**See also**

• `medium-transform`

• `sheet-edges`

• `sheet-position`

• `sheet-size`

**sheet-transform-setter** Generic function

Sets the transform associated with the specified sheet.

**Signature** `sheet-transform-setter transform sheet => transform`

**Parameters**

• `transform` – An instance of type `<transform>`.

• `sheet` – An instance of type `<sheet>`.

**Values**

• `transform` – An instance of type `<transform>`.
Discussion  Sets or modifies the transform associated with sheet.

See also
- `medium-transform-setter`

**sheet-withdrawn?**  Generic function
Returns true if the specified sheet has been withdrawn from the display.

**Signature**  `sheet-withdrawn? sheet => withdrawn?`

**Parameters**
- `sheet` – An instance of type `<sheet>`.

**Values**
- `withdrawn?` – An instance of type `<boolean>`.

Discussion  Returns true if `sheet` has been withdrawn from the display, and is no longer mapped.

See also
- `sheet-mapped?`

**$shift-key**  Constant
A constant that represents the SHIFT key on the keyboard.

**Type**  `<integer>`

**Value**  `ash(1, %modifier_base + 0)`;

Discussion  A constant that represents the SHIFT key on the keyboard.

See also
- `$alt-key`
- `$control-key`
- `$hyper-key`
- `$meta-key`
- `modifier-key-index`
- `modifier-key-index-name`
- `$modifier-keys`
- `$option-key`
- `$super-key`

**$super-key**  Constant
A constant that represents the SUPER key on the keyboard.

**Type**  `<integer>`

**Value**  `ash(1, %modifier_base + 3)`;

Discussion  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 `$option-key` is bound to this constant.

See also
- `$alt-key`
- `$control-key`
• $hyper-key
• $meta-key
• modifier-key-index
• modifier-key-index-name
• $modifier-keys
• $option-key
• $shift-key

**synchronize-display** Generic function
Synchronizes all displays on which the specified drawable is mapped.

**Signature**  
synchronize-display drawable => ()

**Parameters**

• **drawable** – An instance of type type-union(<sheet>, <medium>).

**Discussion**  
Synchronizes all displays on which the specified drawable is mapped.

**text-size** Generic function
Returns information about the size of the specified text on the specified medium.

**Signature**  
text-size medium text #key text-style start end do-newlines? => largest-x largest-y cursor-x cursor-y baseline

**Parameters**

• **medium** – An instance of type <medium>.
• **text** – An instance of type type-union(<string>, <character>).
• **text-style** – An instance of type <text-style>.
• **start** – An instance of type <integer>. Default value: 0.
• **end** – An instance of type <integer>. Default value: size(text).
• **do-newlines?** – An instance of type <boolean>. Default value: #f.
• **do-tabs?** – An instance of type <boolean>. Default value: #f.

**Values**

• **largest-x** – An instance of type <integer>.
• **total-height** – An instance of type <integer>.
• **last-x** – An instance of type <integer>.
• **last-y** – An instance of type <integer>.
• **baseline** – An instance of type <integer>.

**Discussion**

Returns information about the size of text on medium.

If **text-style** is specified, then the information that **text-size** returns is based on the text style it describes.

If **start** and **end** are specified, then these values represent a portion of the string specified by **text**, and only the characters they represent are examined by **text-size**. Both **start** and **end** represent the index of each character in **text**, starting at 0. By default, the whole of **text** is examined.
The `do-newlines?` and `do-tabs?` arguments let you specify how newline or tab characters in text should be handled. If either of these arguments is true, then any newline or tab characters in text are examined, as appropriate. By default, newline characters are ignored.

**text-style-mapping** Generic function

Returns the mapping for the specified text style on the specified port.

**Signature**

```
text-style-mapping port text-style #key character-set => font
```

**Parameters**

- `port` – An instance of type `<port>`.
- `text-style` – An instance of type `<text-style>`.
- `character-set` – An instance of type `<object>`.

**Values**

- `font` – An instance of type `<object>`.

**Discussion**

Returns the mapping for `text-style` on `port`. 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 `text-style` are mapped onto fonts available on the display.

If `character-set` is specified, then this character set is used instead of the default. This is most useful for non-English displays.

**See also**

- `text-style-mapping-exists?`
- `text-style-mapping-setter`

**text-style-mapping-exists?** Generic function

Returns true if a mapping exists for the specified text style on the specified port.

**Signature**

```
text-style-mapping-exists? port text-style #key character-set exact-size? => boolean
```

**Parameters**

- `port` – An instance of type `<port>`.
- `text-style` – An instance of type `<text-style>`.
- `character-set` – An instance of type `<object>`.

**Values**

- `boolean` – An instance of type `<boolean>`.

**Discussion**

Returns true if a mapping exists for `text-style` on `port`. 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**

- `text-style-mapping`
- `text-style-mapping-setter`
text-style-mapping-setter Generic function
Sets the mapping for the specified text style on the specified port.

Signature  text-style-mapping-setter font port text-style #key character-set => font

Parameters
- font – An instance of type <object>.
- port – An instance of type <port>.
- text-style – An instance of type <text-style>.
- character-set – An instance of type <object>.

Values
- font – An instance of type <object>.

Discussion
Sets the mapping for text-style on port to the specified font. 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 text-style are mapped onto fonts available on the display.

If character-set is specified, then this character set is used instead of the default. This is most useful for non-English displays.

See also
- text-style-mapping
- text-style-mapping-exists?
- <undefined-text-style-mapping>
Superclasses  <error>

Discussion  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

See also
• text-style-mapping
• text-style-mapping-exists?
• text-style-mapping-setter

<window-configuration-event> Instantiable Sealed Class
The class of events involving changes to the window configuration.

Superclasses  <window-event>

Discussion  The class of events involving changes to the window configuration.

Operations

See also
• <window-repaint-event>

>window-event> Open Abstract Class
The base class of events that occur in windows.

Superclasses  <sheet-event>

Init-Keywords

• region – An instance of type <region>. Required.

Discussion  The base class of events that occur in windows. Two types of event can occur:
• Changes to the configuration of the window.
• Changes that require the window to be repainted.

The region: init-keyword specifies a region in which the event occurs.

Operations

The following operation is exported from the DUIM-Sheets module.

• event-region

See also
• event-region
• <window-configuration-event>
• <window-repaint-event>

>window-repaint-event> Instantiable Sealed Class
The class of events involving repainting of a window.

Superclasses  <window-event>

Discussion  The class of events involving repainting of a window.
Operations

See also

- `handle-repaint`
- `queue-repaint`
- `repaint-sheet`
- `<window-configuration-event>`

**with-brush** Macro

Executes the supplied code using the specified brush characteristics.

**Macro Call**

```dylan
with-brush ({medium} #rest {brush-initargs}*) {body} end
```

**Parameters**

- `medium` – A Dylan body*bnf*.
- `brush-initargs` – Dylan arguments*bnf*.
- `body` – A Dylan body*bnf*.

**Discussion**

Executes `body` using the brush characteristics specified by `brush-initargs`, and applies the results to `medium`. The `medium` specified should be an instance of type `<medium>`. The `brush-initargs` can be any valid arguments that specify an instance of `<brush>`.

See also

- `with-pen`

**with-clipboard** Macro

Evaluates a body of code with a clipboard grabbed.

**Macro Call**

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

**Parameters**

- `clipboard` – A Dylan variable-name*bnf*.
- `sheet` – A Dylan variable-name*bnf*.
- `body` – A Dylan body*bnf*.

**Values**

- `values` – Instances of `<object>`.

**Discussion**

Evaluates `body` with the clipboard grabbed, returning the results to the clipboard.

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

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

You can add more than one format of your data to the clipboard within the scope of 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
with-clipping-region Macro
Executes the supplied code using the specified clipping region.

**Macro Call**
```
with-clipping-region ({medium} {region} {body} end
```

**Parameters**
- `medium` – A Dylan expression*bnf*.
- `region` – A Dylan expression*bnf*.
- `body` – A Dylan body*bnf*.

**Discussion**
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 Macro
Executes the supplied code using the specified cursor settings for a sheet.

**Macro Call**
```
with-cursor-visible ({sheet} {visible?} {body} end
```

**Parameters**
- `sheet` – A Dylan expression*bnf*.
- `visible?` – A Dylan expression*bnf*.
- `body` – A Dylan body*bnf*.

**Discussion**
Executes `body` on the specified `sheet`. If `visible?` is true, then the pointer cursor associated with `sheet` is visible throughout the operation. If `visible?` is false, then the pointer cursor is hidden.

The expression `sheet` should evaluate to an instance of `<sheet>`. The expression `visible?` should evaluate to a boolean value.

with-drawing-options Macro
Runs a body of code in the context of a set of drawing options.

**Macro Call**
```
with-drawing-options ({medium} #rest {options}*) {body} end
```

**Parameters**
- `medium` – A Dylan expression*bnf*.
- `options` – Dylan arguments*bnf*.
- `body` – A Dylan body*bnf*.

**Discussion**
Runs a body of code in the context of a set of drawing options. The options specified are passed to the function `do-with-drawing-options` for execution.

The `medium` expression should evaluate to an instance of `<medium>`.

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 withdrawing-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
end

Example

with-drawing-options (medium, brush: $red)
  draw-rectangle (medium, 0, 0, 100, 200, filled?: #t)
end

See also
  • do-with-drawing-options
  • with-sheet-medium

withdraw-sheet Generic function
With draws the specified sheet from the current display.

Signature  withdraw-sheet sheet => ()

Parameters
  • sheet – An instance of type <sheet>.

Discussion  Withdraws the specified sheet from the current display.

with-frame-manager Macro
Executes the supplied code in the context of the specified frame manager.

Macro Call  with-frame-manager ({framem}) {body} end

Parameters
  • framem – A Dylan expression*bnf*.
  • body – A Dylan body*bnf*.

Discussion  Executes body in the context of framem, by dynamically binding the expression framem to *current-frame-manager*.

In practice, you do not need to use with-frame-manager unless you are certain that your code needs to run on a non-primary frame manager.

The main place where you need to use this macro is when you call make to create a gadget outside of one of the pane or layout clauses in define frame.

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.

See also
  • <frame-manager>

with-identity-transform Macro
Executes the supplied code while retaining the current transform.
**with-identity-transform**

**Macro Call**

\[
\text{with-identity-transform} (\{ \text{medium} \}) \{ \text{body} \} \end
\]

**Parameters**

- **medium** – A Dylan expression*bnf*.
- **body** – A Dylan body*bnf*.

**Discussion**

Executes \textit{body} while retaining the current transform for \textit{medium}.

The \textit{medium} expression should evaluate to an instance of \textit{<medium>}.  

**with-pen**

**Macro**

Executes the supplied code using the specified pen characteristics.

**Macro Call**

\[
\text{with-pen} (\{ \text{medium} \} \#\text{rest} \{ \text{pen-initargs} \} \*) \{ \text{body} \} \end
\]

**Parameters**

- **medium** – A Dylan expression*bnf*.
- **pen-initargs** – Dylan arguments*bnf*.
- **body** – A Dylan body*bnf*.

**Discussion**

Executes \textit{body} using the pen characteristics specified by \textit{pen-initargs}, and applies the results to the expression \textit{medium}.

The \textit{medium} specified should be an instance of type \textit{<medium>}. The \textit{pen-initargs} can be any valid arguments that specify an instance of \textit{<pen>}.  

**See also**

- **with-brush**

**with-pointer-grabbed**

**Macro**

Executes a body of code, forwarding all pointer events to a sheet.

**Macro Call**

\[
\text{with-pointer-grabbed} (\{ \text{sheet} \} \#\text{rest} \{ \text{options} \} \*) \{ \text{body} \} \end
\]

**Parameters**

- **sheet** – A Dylan expression*bnf*.
- **options** – Dylan arguments*bnf*.
- **body** – A Dylan body*bnf*.

**Discussion**

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

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

**See also**

- **do-with-pointer-grabbed**

**with-rotation**

**Macro**

Executes a body of code with a specified rotation.

**Macro Call**

\[
\text{with-rotation} (\{ \text{medium} \} \{ \text{angle} \} \) \{ \text{body} \} \end
\]
Parameters

- **medium** – A Dylan expression*bnf*.
- **angle** – A Dylan argument*bnf*.
- **body** – A Dylan body*bnf*.

Discussion

Executes a body of code with a specified rotation. The rotation occurs within the expression `medium`. This macro calls `with-transform` to perform the rotation.

The `medium` specified should be an instance of type `<medium>`. The `angle` should evaluate to an instance of type `<real>`.

See also

- `with-scaling`
- `with-transform`
- `with-translation`

`with-scaling` Macro

Executes a body of code with a specified scaling.

**Macro Call**  
`with-scaling ({ medium } { scale-x } { scale-y }) { body } end`

**Parameters**

- **medium** – A Dylan expression*bnf*.
- **scale-x** – A Dylan argument*bnf*.
- **scale-y** – A Dylan argument*bnf*.
- **body** – A Dylan body*bnf*.

Discussion

Executes a body of code with a specified scaling, denoted by `scale-x` and `scale-y`. The scaling occurs within the expression `medium`. This macro calls `with-transform` to perform the scaling.

The `medium` specified should be an instance of type `<medium>`. The `scale-x` and `scale-y` should evaluate to an instance of type `<real>`.

See also

- `with-rotation`
- `with-transform`
- `with-translation`

`with-sheet-medium` Macro

 Associates a sheet with a medium.

**Macro Call**  
`with-sheet-medium (medium = sheet ) { body } end`

**Parameters**

- **medium** – A Dylan name*bnf*.
- **sheet** – A Dylan expression*bnf*.
- **body** – A Dylan body*bnf*.
Discussion

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

```dylan
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*
- *with-drawing-options*

### with-text-style Macro

Runs a body of code in the context of a text style.

**Macro Call**

```dylan
with-text-style ({medium} #rest {style-initargs} *) {body} end
```

**Parameters**

- **medium** – A Dylan expression*bnf*.
- **style-initargs** – Dylan arguments*bnf*.
- **body** – A Dylan body*bnf*.

**Discussion**

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*

### with-transform Macro

Executes a body of code with a specified transform.
Macro Call  with-transform (medium {transform} body) end

Parameters

- **medium** – A Dylan expression*bnf*.
- **transform** – A Dylan expression*bnf*.
- **body** – A Dylan body*bnf*.

Discussion

Executes a body of code with a specified `transform`. The transform occurs within `medium`. This macro is used by `with-rotation`, `with-scaling`, and `with-translation`, and calls methods for `do-with-transform`.

The `medium` specified should be an instance of type `<medium>`. The `transform` specified should be an instance of type `<transform>`.

See also

- `do-with-transform`
- `with-rotation`
- `with-scaling`
- `with-translation`

**with-translation Macro**

Executes a body of code with a specified translation.

Macro Call  with-translation (medium {dx} {dy} body) end

Parameters

- **medium** – A Dylan expression*bnf*.
- **dx** – A Dylan argument*bnf*.
- **dy** – A Dylan argument*bnf*.
- **body** – A Dylan body*bnf*.

Discussion

Executes a body of code with a specified translation, denoted by `dx` and `dy`. The translation occurs within `medium`. This macro calls `with-transform` to perform the translation.

The `medium` specified should be an instance of type `<medium>`. The `dx` and*dy* should evaluate to an instance of type `<real>`.

See also

- `with-rotation`
- `with-scaling`
- `with-transform`
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. DUIM-Graphics Module 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:

A text/graphics layer in which you specify the desired visual appearance independent of device resolution and characteristics

1. 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.

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.

- **Sheets** 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.

- **Streams** 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).

- 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.

- For more general information about DUIM streams, you should refer to the manual *Library Reference: System and I/O*.

### 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.

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 a 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 *How pixels are defined to be “inside” and “outside” shapes.*
• 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.

![Diagram of inside and outside pixels]

**Fig. 7.1**: 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 *Two triangles* 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.

![Diagram of two triangles]

**Fig. 7.2**: 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.

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 *Two forms of a circle inscribed in a square*, 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.

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 *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
Fig. 7.3: Choosing any two of the shaded pixels causes asymmetry

Fig. 7.4: Two forms of a circle inscribed in a square

Fig. 7.5: An aesthetically pleasing circle
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.

**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 *Two examples of lines of thickness 1* 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.

![Fig. 7.6: Two examples of lines of thickness 1](image1)

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 *Two examples of lines of thickness 2* are both reasonable.

![Fig. 7.7: Two examples of lines of thickness 2](image2)

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 *Two possible definitions of horizontal lines. Left figure is X11 definition*). 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.

**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 *DUIM-Graphics Module*.

### 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.

#### start-path Generic function

**Signature**  
`start-path drawable => ()`

**Discussion**

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

**Signature**  
`end-path drawable => ()`

**Discussion**

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

**Signature**  
`close-path drawable => ()`

**Discussion**

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**

**Signature**

```lisp
abort-path drawable => ()
```

**Discussion**

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**

**Signature**

```lisp
fill-path drawable => ()
```

**Discussion**

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**

**Signature**

```lisp
stroke-path drawable => ()
```

**Discussion**

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.

### 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**

**Signature**

```lisp
line-to drawable x y => ()
```

**Discussion**

Draws a line from the current position in the path to `(x, y)`.

**curve-to Generic function**

**Signature**

```lisp
curve-to drawable x1 y1 x2 y2 x3 y3 => ()
```

**Discussion**

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**

**Signature**

```lisp
move-to drawable x y => ()
```

**Discussion**

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**

**Signature**

```lisp
arc-to drawable center-x center-y radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key
start-angle end-angle => ()
```

**Discussion**

Draws an arc in the current path on `drawable`.

Description of the arguments for arc-to
The center of the arc is defined by \((\text{center-x}, \text{center-y})\), the points furthest away from the center for each radius are calculated by adding \(\text{radius-1-dx}\) and \(\text{radius-1-dy}\) to \(\text{center-x}\) and \(\text{center-y}\) respectively (to calculate the outermost points for the first radius), and adding \(\text{radius-2-dx}\) and \(\text{radius-2-dy}\) to \(\text{center-x}\) and \(\text{center-y}\) respectively (to calculate the outermost points for the second radius).

The arguments \(\text{start-angle}\) and \(\text{end-angle}\) 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, \(\text{line-to}\) performs the same operation as \(\text{line-to}\), 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.

**DUIM-Graphics Module**

This section contains a complete reference of all the interfaces that are exported from the \textit{duim-graphics} module.

**Abort-path** Generic function

Aborts the current path on the specified drawable object.

**Signature**

\[
\text{abort-path} \quad \text{drawable} \Rightarrow ()
\]

**Parameters**

- \textit{drawable} – An instance of type \texttt{type-union(<sheet>, <medium>)}.

**Discussion**

Aborts the current path on \textit{drawable}. Any operations that have been performed since the last call to \textit{start-path} are discarded.

**See also**

- \textit{close-path}
- \textit{end-path}
- \textit{start-path}

**Arc-to** Generic function

Draws an arc in the current path on the specified drawable.

**Signature**

\[
\text{arc-to} \quad \text{drawable \ center-x \ center-y \ radius-1-dx \ radius-1-dy \ radius-2-dx \ radius-2-dy \ #key \ start-angle \ end-angle} \Rightarrow ()
\]

**Signature**

\[
\text{arc-to*} \quad \text{drawable \ center \ radius-1-dx \ radius-1-dy \ radius-2-dx \ radius-2-dy \ #key \ start-angle \ end-angle} \Rightarrow ()
\]
Parameters

- **drawable** – An instance of type type-union(<sheet>, <medium>).
- **radius-1-dx** – An instance of type <real>.
- **radius-1-dy** – An instance of type <real>.
- **radius-2-dx** – An instance of type <real>.
- **radius-2-dy** – An instance of type <real>.
- **start-angle** – An instance of type false-or(<real>).
- **end-angle** – An instance of type false-or(<real>).

The following arguments are specific to `arc-to`.

Parameters

- **center-x** – An instance of type <real>.
- **center-y** – An instance of type <real>.

The following argument is specific to `arc-to*`.

Parameters

- **center** – An instance of type <transform>.

Discussion

Draws an arc in the current path on the specified drawable.

This function is used, in combination with `line-to`, `curve-to`, and `move-to`, to define a path. The function `start-path` should be used to start the definition of the path, and `end-path` can be used to finish the definition.

The center of the arc is defined by (center-x, center-y), 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 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 the radius vectors radius-2-dx and radius-2-dy to center-x and center-y respectively (to calculate the outermost points for the second radius).

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`
- `draw-bezier-curve`
- `draw-line`
- `line-to`
- `move-to`

**close-path**

Generic function
Closes the current path on the specified drawable.

**Signature**

`close-path drawable => ()`

**Parameters**

- `drawable` – An instance of type `type-union(<sheet>, <medium>)`.

**Discussion**

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.

Only closed paths can be filled, although `fill-path` will close a non-closed path automatically.

See also

- `abort-path`
- `end-path`
- `start-path`

**copy-area**

Generic function
Copies a rectangle of pixels from a specified medium to the same medium.

**Signature**

`copy-area medium from-x from-y width height to-x to-y #key function => ()`

**Parameters**

- `medium` – An instance of type `<medium>`.
- `from-x` – An instance of type `<coordinate>`.
- `from-y` – An instance of type `<coordinate>`.
- `width` – An instance of type `<integer>`.
- `height` – An instance of type `<integer>`.
- `to-x` – An instance of type `<coordinate>`.
- `to-y` – An instance of type `<coordinate>`.
Discussion  Copies the pixels from the medium starting at the position specified by \((\text{from-}x, \text{from-}y)\) to the position \((\text{to-}x, \text{to-}y)\) on the same medium. A rectangle whose width and height is specified by \(\text{width}\) and \(\text{height}\) is copied. If medium is a medium or a stream, then the x and y values are transformed by the user transformation. The copying must be done by \text{medium-copy-copy}.

See also
- \text{copy-from-pixmap}
- \text{copy-to-pixmap}

\text{copy-from-pixmap} Generic function
Copies a rectangle of pixels from the specified pixmap to the specified medium.

Signature  \text{copy-from-pixmap} \text{ pixmap pixelmap-x pixelmap-y width height medium medium-x medium-y} #key function => ()

Parameters
- \text{pixelmap} – An instance of type \text{<pixmap>}. 
- \text{pixelmap-x} – An instance of type \text{<coordinate>}. 
- \text{pixelmap-y} – An instance of type \text{<coordinate>}. 
- \text{width} – An instance of type \text{<integer>}. 
- \text{height} – An instance of type \text{<integer>}. 
- \text{medium} – An instance of type \text{<coordinate>}. 
- \text{medium-x} – An instance of type \text{<coordinate>}. 
- \text{medium-y} – An instance of type \text{<coordinate>}. 
- \text{function} – An instance of type \text{<function>}. Default value: \$boole-1.

Discussion  Copies a rectangle of pixels from \text{pixelmap} starting at the position specified by \((\text{pixelmap-x, pixelmap-y})\) into \text{medium} at the position \((\text{medium-x, medium-y})\). A rectangle whose width and height is specified by \(\text{width}\) and \(\text{height}\) is copied. If \text{medium} is a medium or a stream, then \text{medium-x} and \text{medium-y} are transformed by the user transformation. The copying must be done by \text{medium-copy-copy}.

See also
- \text{copy-area}
- \text{copy-to-pixmap}
- \text{<pixmap>}

\text{copy-to-pixmap} Generic function
Copies a rectangle of pixels from the specified medium to the specified pixmap.

Signature  \text{copy-to-pixmap} \text{ medium medium-x medium-y width height pixelmap pixelmap-x pixelmap-y} #key function => ()

Parameters
- \text{medium} – An instance of type \text{<medium>}. 
- \text{medium-x} – An instance of type \text{<coordinate>}. 
- \text{medium-y} – An instance of type \text{<coordinate>}. 
- \text{width} – An instance of type \text{<integer>}. 
- \text{height} – An instance of type \text{<integer>}. 

230 Chapter 7. DUIM-Graphics Library
• **pixmap** – An instance of type `<pixmap>`.

• **pixmap-x** – An instance of type `<coordinate>`.

• **pixmap-y** – An instance of type `<coordinate>`.

• **function** – An instance of type `<function>`. Default value: `$boole-1`.

Discussion

Copies the pixels from the medium starting at the position specified by (medium-x, medium-y) into pixmap at the position specified by (pixmap-x, pixmap-y). A rectangle whose width and height is specified by width and height is copied. If medium is a medium or a stream, then medium-x and medium-y are transformed by the user transformation. The copying must be done by medium-copy-copy.

If pixmap is not supplied, a new pixmap will be allocated.

See also

• copy-area

• copy-from-pixmap

curve-to Generic function

Draws a curve through three specified points in the current path on the specified drawable.

**Signature**  
`curve-to drawable x1 y1 x2 y2 x3 y3 => ()`

**Signature**  
`curve-to* drawable point1 point2 point3 => ()`

Parameters

• **drawable** – An instance of type type-union(<sheet>, <medium>).

The following arguments are specific to curve-to.

Parameters

• **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 curve-to*.

Parameters

• **point1** – An instance of type `<transform>`.

• **point2** – An instance of type `<transform>`.

• **point3** – An instance of type `<transform>`.

Discussion

Draws a curve in the current path on drawable starting from the current position, and passing through (x1, y1), (x2, y2), and (x3, y3).

This function is used, in combination with :gf:line-to', move-to, and arc-to, to define a path. The function start-path should be used to start the definition of the path, and end-path can be used to finish the definition.
The function curve-to* is identical to curve-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
- draw-bezier-curve
- draw-line
- line-to
- move-to

**destroy-pixmap** Generic function

Destroys the specified pixmap.

**Signature**

destroy-pixmap pixmap => ()

**Parameters**

- **pixmap** – An instance of type <pixmap>.

**Discussion**

Destroys **pixmap**.

See also

- draw-pixmap

**do-with-output-to-pixmap** Generic function

Returns a pixmap for the specified medium.

**Signature**

do-with-output-to-pixmap medium continuation #key width height clear? => pixmap

**Parameters**

- **medium** – An instance of type <medium>.
- **continuation** – An instance of type <function>.
- **width** – An instance of type <integer>.
- **height** – An instance of type <integer>.
- **clear?** – An instance of type <boolean>. Default value: #t.

**Values**

- **pixmap** – An instance of type <pixmap>.

**Discussion**

Returns a pixmap for the specified medium. This function is called by **with-output-to-pixmap** and returns the pixmap that is operated on. If you are subclassing ‘class:<medium>‘, you must define new methods on this function.

The **width** and **height** 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 **with-output-to-pixmap**.

See also

- **with-output-to-pixmap**

**draw-arrow** Generic function

Draws an arrow between two specified points.
Signature draw-arrow drawable x1 y1 x2 y2 #key from-head? to-head? head-length head-width => ()

Signature draw-arrow* drawable point1 point2 #key from-head? to-head? head-length head-width => ()

Parameters
- drawable – An instance of type type-union(<sheet>, <medium>).
- head-length – An instance of type <integer>. Default value: 10.
- head-width – An instance of type <integer>. Default value: 5.

The following arguments are specific to draw-arrow.

Parameters
- 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-arrow*.

Parameters
- point1 – An instance of type <transform>.
- point2 – An instance of type <transform>.

Discussion
Draws an arrow on drawable between two (x1, y1) and (x2, y2), using the current pen. Dashed lines start dashing from the first point.

If from-head? is #t, then the arrow-head points from (x1, y1) to (x2, y2). If to-head? is #t, then the arrow-head points from (x2, y2) to (x1, y1).

If both from-head? and to-head? are #t, then a double-headed arrow is drawn.

The arguments head-length and head-width specify the length and width of the arrow-head respectively, in pixels.
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`

**draw-bezier-curve** Generic function

Draws a bezier curve through the specified set of points.

*Signature* `draw-bezier-curve` `sheet coord-seq #key filled? => ()`

*Signature* `draw-bezier-curve*` `drawable points #key filled? => ()`

*Parameters*


The following arguments are specific to `draw-bezier-curve`.

*Parameters*

- `sheet` – An instance of type `<sheet>`.
- `coord-seq` – An instance of type `limited(<sequence>, of: <coordinate>)`.

The following arguments are specific to `draw-bezier-curve*`.

*Parameters*

- `drawable` – An instance of type `type-union(<sheet>, <medium>)`.
- `points` – An instance of type `limited(<sequence>, of: <point>)`.

*Discussion*

Draws a bezier curve on `sheet` or `drawable` (depending on the function you use) through the sequence of coordinates given by `coord-seq`, using the current pen. Dashed lines start dashing from the first point.

If `filled?` is `#t` then the bezier-curve will be filled, using the current brush.

The function `draw-bezier-curve*` is identical to `draw-bezier-curve`, 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`
- `draw-line`

**draw-circle** Generic function

Draws a circle with the specified center and radius.

*Signature* `draw-circle` `drawable center-x center-y radius #key start-angle end-angle filled? => ()`

*Signature* `draw-circle*` `drawable center radius #key start-angle end-angle filled? => ()`

*Parameters*

- `drawable` – An instance of type `type-union(<sheet>, <medium>)`.
- `radius` – An instance of type `<real>`.
- `start-angle` – An instance of type `false-or(<real>)`. 
• **end-angle** – An instance of type `false-or(<real>)`.

• **filled?** – An instance of type `<boolean>`. Default value: `#t`.

The following arguments are specific to `draw-circle`.

**Parameters**

• **center-x** – An instance of type `<real>`.

• **center-y** – An instance of type `<real>`.

The following argument is specific to `draw-circle*`.

**Parameters**

• **center** – An instance of type `<transform>`.

**Discussion**

Draws a circle on `drawable` with center `(center-x, center-y)` and a radius of `radius` pixels, using the current pen.

The `start-angle` and `end-angle` arguments let you draw a sector of a circle rather than a whole circle.

If `filled?` is `#t`, then the circle will be filled, using the current brush.

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`

• `draw-oval`

**draw-ellipse** Generic function

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? => ()
```

**Signature**

```
draw-ellipse* drawable center radius-1-dx radius-1-dy radius-2-dx radius-2-dy #key start-angle end-angle filled? => ()
```

**Parameters**

• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.

• **radius-1-dx** – An instance of type `<real>`.

• **radius-1-dy** – An instance of type `<real>`.

• **radius-2-dx** – An instance of type `<real>`.

• **radius-2-dy** – An instance of type `<real>`.

• **start-angle** – An instance of type `false-or(<real>)`.

• **end-angle** – An instance of type `false-or(<real>)`.

• **filled?** – An instance of type `<boolean>`. Default value: `#t`.

The following arguments are specific to `draw-ellipse`.

**Parameters**
• **center-x** – An instance of type `<real>`.
• **center-y** – An instance of type `<real>`.

The following argument is specific to `draw-ellipse*`.

**Parameters**

• **center** – An instance of type `<transform>`.

**Discussion**

Draws an ellipse on `drawable` with the specified center and extreme points, using the current pen.

The center of the ellipse is defined by `(center-x, center-y)`, 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 `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 the radius vectors `radius-2-dx` and `radius-2-dy` to `center-x` and `center-y` respectively (to calculate the outermost points for the second radius).

Please note that `draw-ellipse` does not currently support non-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` 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`
• `draw-oval`

**draw-image** Generic function

Draws the specified image at the specified position.

**Signature**  
`draw-image drawable image x y => ()`

**Signature**  
`draw-image* drawable image point => ()`

**Parameters**

• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.
• **image** – An instance of type `<image>`.
The following arguments are specific to draw-image.

**Parameters**
- **x** – An instance of type `<real>`.
- **y** – An instance of type `<real>`.

The following argument is specific to draw-image*.

**Parameters**
- **point** – An instance of type `<transform>`.

**Discussion**
Draws image on drawable at (x, y).

The function draw-image* is identical to draw-image, 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-pixmap
- draw-text

**draw-line** Generic function
Draws a line between the specified points.

**Signature**

```
(draw-line drawable x1 y1 x2 y2) => ()
```

**Signature**

```
(draw-line* drawable point1 point2) => ()
```

**Parameters**
- **drawable** – An instance of type `type-union(<sheet>, <medium>)`.

The following arguments are specific to draw-line.

**Parameters**
- **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*.

**Parameters**
- **point1** – An instance of type `<transform>`.
- **point2** – An instance of type `<transform>`.

**Discussion**
Draws a line on drawable between (x1, y1) and (x2, y2), 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**
draw-lines Generic function
Draws a series of lines between the specified sequence of points.

Signature  draw-lines drawable coord-seq => ()
Signature  draw-lines* drawable points => ()

Parameters
- **drawable** – An instance of type type-union(<sheet>, <medium>).
The following argument is specific to draw-lines.

Parameters
- **coord-seq** – An instance of type limited(<sequence>, of: <coordinate>).
The following argument is specific to draw-lines*.

Parameters
- **points** – An instance of type limited(<sequence>, of: <point>).

Discussion
Draws a series of lines on drawable 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**
- **draw-points**
- **draw-rectangles**

draw-oval Generic function
Draws an oval with the specified center and radii.

Signature  draw-oval drawable center-x center-y x-radius y-radius #key filled? => ()
Signature  draw-oval* drawable center x-radius y-radius #key filled? => ()
Parameters

- **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`.

Parameters

- **center-x** – An instance of type `<real>`.
- **center-y** – An instance of type `<real>`.

The following argument is specific to `draw-oval*`.

Parameters

- **center** – An instance of type `<transform>`.

Discussion

Draws an oval on `drawable` with center `(center-x, center-y)` and radii defined by `x-radius` and `y-radius`, using the current pen.

Ovals are similar to ellipses, except that they have straight edges.

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

- `draw-circle`
- `draw-ellipse`

**draw-pixmap** Generic function

Draws the contents of the specified pixmap at the specified point.

**Signature**

```
draw-pixmap drawable pixmap x y #key function => ()
```

**Signature**

```
draw-pixmap* drawable pixmap point #key function => ()
```

**Parameters**
• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.
• **pixmap** – An instance of type `<pixmap>`.
• **function** – An instance of type `<function>`. Default value: `$boole-1`.

The following arguments are specific to `draw-pixmap`.

**Parameters**

• **x** – An instance of type `<real>`.
• **y** – An instance of type `<real>`.

The following argument is specific to `draw-pixmap*`.

**Parameters**

• **point** – An instance of type `<transform>`.

**Discussion**

Draws the contents of `pixmap` on `drawable` at (x, y).

The function `draw-pixmap*` is identical to `draw-pixmap`, 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**

• `destroy-pixmap`
• `draw-image`
• `draw-text`
• `make-pixmap`

**draw-point** Generic function

Draws a single point at the specified position.

**Signature** `draw-point` `drawable x y => ()`

**Signature** `draw-point*` `drawable point => ()`

**Parameters**

• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.

The following arguments are specific to `draw-point`.

**Parameters**

• **x** – The x coordinate.
• **y** – The y coordinate.

The following argument is specific to `draw-point*`.

**Parameters**

• **point** – An instance of type `<transform>`.

**Discussion**

Draws a single point on `drawable` at (x, y).

The function `draw-point*` is identical to `draw-point`, 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`
- `draw-points`

**draw-points** Generic function

Draws a sequence of points at the specified positions.

**Signature**
draw-points drawable coord-seq => ()

draw-points* drawable points => ()

**Parameters**

- **drawable** – An instance of type `type-union(<sheet>, <medium>)`.

The following argument is specific to `draw-points`.

**Parameters**

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

The following argument is specific to `draw-points*`.

**Parameters**

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

**Discussion**

Draws a sequence of points on `drawable` at the specified positions.

The function `draw-points*` is identical to `draw-points`, 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-lines`
- `draw-point`
- `draw-rectangles`

**draw-polygon** Generic function

Draws a polygon joining the specified points.

**Signature**
draw-polygon drawable coord-seq #key closed? filled? => ()

draw-polygon* drawable points #key closed? filled? => ()

**Parameters**

- **drawable** – An instance of type `type-union(<sheet>, <medium>)`.

- **closed?** – An instance of type `<boolean>`. Default value: `#t`.

- **filled?** – An instance of type `<boolean>`. Default value: `#t`.

The following argument is specific to `draw-polygon`.

**Parameters**

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

The following argument is specific to `draw-polygon*`.  

7.6. DUIM-Graphics Module
Parameters

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

Discussion

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`
- `draw-regular-polygon`
- `draw-triangle`

**draw-rectangle** Generic function

Draws a rectangle at the specified position.

**Signature** `draw-rectangle drawable x1 y1 x2 y2 #key filled? => ()`

**Signature** `draw-rectangle* drawable point1 point2 #key filled? => ()`

Parameters

- `drawable` – An instance of type `type-union(<sheet>, <medium>)`.

The following arguments are specific to `draw-rectangle`.

Parameters

- `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-rectangle*`.

Parameters

- `point1` – An instance of type `<transform>`.
- `point2` – An instance of type `<transform>`.

Discussion

Draws a rectangle on `drawable` with left and right corners at `(x1, y1)` and `(x2, y2)`, using the current pen. Dashed lines start dashing at the starting point of the first segment.

Note that the specified points could represent either top or bottom corners: only one rectangle is possible between and pair of points.

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`
- `draw-rectangles`
- `draw-regular-polygon`
- `draw-triangle`

`draw-rectangles` Generic function

Draws a sequence of rectangles at the specified positions.

Signature `draw-rectangles` `drawable coord-seq` #key filled? => ()

Signature `draw-rectangles*` `drawable points` #key filled? => ()

Parameters

- `drawable` – An instance of type `type-union(<sheet>, <medium>)`.

The following argument is specific to `draw-rectangles`.

Parameters

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

The following argument is specific to `draw-rectangles*`.

Parameters

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

Discussion

Draws a sequence of rectangles on `drawable` 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.

If `filled?` is #t then the rectangles will be filled, using the current brush.

The function `draw-rectangles*` is identical to `draw-rectangles`, 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-lines`
• **draw-points**

• **draw-rectangle**

### **draw-regular-polygon** Generic function

Draws a regular polygon that touches the specified points, and has the specified number of sides.

**Signature**  
draw-regular-polygon drawable x1 y1 x2 y2 nsides #key handedness closed? filled? => ()

**Signature**  
draw-regular-polygon* drawable point1 point2 nsides #key handedness closed? filled? => ()

**Parameters**

- **drawable** – An instance of type *type-union(<sheet>, <medium>)*.
- **nsides** – An instance of type *<integer>*.
- **handedness** – Default value: #"left".
- **closed?** – An instance of type *<boolean>*. Default value: #t.
- **filled?** – An instance of type *<boolean>*. Default value: #t.

The following arguments are specific to **draw-regular-polygon**.

**Parameters**

- **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**.*.

**Parameters**

- **point1** – An instance of type *<transform>*.
- **point2** – An instance of type *<transform>*.

**Discussion**

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.

![Regular Polygon](image)

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`
- `draw-rectangle`
- `draw-triangle`

**draw-text Generic function**

Draws text at the specified point, in a specified direction.

**Signature**

`draw-text drawable text x y #key start end align-x align-y towards-point transform-glyphs? => ()`

**Signature**

`draw-text* drawable text point #key start end align-x align-y towards-point transform-glyphs? => ()`

**Parameters**

- **drawable** – An instance of type `type-union(<sheet>, <medium>)`.
- **text** – An instance of type `type-union(<string>, <character>)`.
- **start** – An instance of type `<integer>`. Default value: 0.
- **end** – An instance of type `<integer>`. Default value: `size(text)`.
- **align-x** – An instance of type `one-of(#"left", #"right", #"center")`. Default value: #"left".
- **align-y** – An instance of type `one-of(#"top", #"bottom", #"baseline")`. Default value: #"baseline".
- **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`.

**Parameters**

- **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*`.

**Parameters**

- **towards-point** – An instance of type `<transform>`.
- **point** – An instance of type `<transform>`.

**Discussion**

Draws text from `text` on `drawable` at `(x, y)`. Text is drawn in the direction of the point `(towards-x, 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 \textit{align-\textit{x}}, the whole of the distance between \((x, y)\) and \((\textit{towards-\textit{x}}, \textit{towards-\textit{y}})\) is used to align text. Thus, if \textit{align-\textit{x}} is \#"right", the text will appear closer to \((\textit{towards-\textit{x}}, \textit{towards-\textit{y}})\) than to \((x, y)\), assuming text occupies less space than the distance between these two points.

The argument \textit{transform-glyphs?} controls whether the text is reversed in cases when \textit{towards-\textit{x}} is less than \(x\). If \textit{transform-glyphs?} is \#t, then text is reversed in these cases, that is, the last character of \textit{text} to be written is still closest to the point \((\textit{towards-\textit{x}}, \textit{towards-\textit{y}})\), and the text appears reversed. If \textit{transform-glyphs?} is \#f, then the first character of \textit{text} to be written is closest to the point \((\textit{towards-\textit{x}}, \textit{towards-\textit{y}})\), and the text does not appear reversed.

If \textit{do-tabs?} is \#t, then any tab characters in \textit{text} are honored, and are drawn as tabs. If \textit{do-tabs?} is \#f, then tab characters are replaced by spaces.

The function \textit{draw-text*} is identical to \textit{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.

\textbf{See also}

- \textit{draw-image}
- \textit{draw-pixmap}

\textbf{draw-triangle} \textbf{Generic function}

Draws a triangle between the specified points.

\textbf{Signature} \textit{draw-triangle} \textit{drawable} \textit{x1 y1 x2 y2 x3 y3} \#key filled? => ()

\textbf{Signature} \textit{draw-triangle*} \textit{drawable} \textit{p1 p2 p3} \#key filled? => ()

\textbf{Parameters}

- \textit{drawable} – An instance of type \texttt{type-union(<sheet>, <medium>)}.
- \textit{filled?} – An instance of type \texttt{<boolean>}. Default value: \#t.

The following arguments are specific to \textit{draw-triangle}.

\textbf{Parameters}

- \textit{x1} – An instance of type \texttt{<real>}.
- \textit{y1} – An instance of type \texttt{<real>}.
- \textit{x2} – An instance of type \texttt{<real>}.
- \textit{y2} – An instance of type \texttt{<real>}.
- \textit{x3} – An instance of type \texttt{<real>}.
- \textit{y3} – An instance of type \texttt{<real>}.

The following arguments are specific to \textit{draw-triangle*}.

\textbf{Parameters}

- \textit{p1} – An instance of type \texttt{<transform>}.
• p2 – An instance of type <transform>.
• p3 – An instance of type <transform>.

Discussion

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
• draw-rectangle
• draw-regular-polygon

end-path Generic function

Ends the definition of the current path in the specified drawable object.

Signature  end-path drawable => ()

Parameters

• drawable – An instance of type type-union(<sheet>, <medium>).

Discussion

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.

See also

• abort-path
• close-path
• start-path

fill-path Generic function

Uses the current brush to fill the current path on the specified drawable object.

Signature  fill-path drawable => ()

Parameters

• drawable – An instance of type type-union(<sheet>, <medium>).
Discussion  Uses the current brush to fill the current path on drawable. If the path has not already been closed using close-path, it is closed automatically.

See also

- stroke-path
- close-path

**line-to** Generic function

Draws a line from the current position in the path to a new position.

*Signature*  line-to drawable x y => ()

*Signature*  line-to* drawable point => ()

**Parameters**

- **drawable** – An instance of type type-union(<sheet>, <medium>).

The following arguments are specific to line-to.

**Parameters**

- **x** – An instance of type <real>.
- **y** – An instance of type <real>.

The following argument is specific to line-to*.

**Parameters**

- **point** – An instance of type <transform>.

Discussion

Draws a line from the current position in the path to (x, y).

This function is used, in combination with move-to, curve-to, and arc-to, to define a path. The function start-path should be used to start the definition of the path, and end-path can be used to finish the definition.

The function line-to* is identical to line-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
- curve-to
- draw-bezier-curve
- draw-line
- move-to

**make-pixmap** Generic function

Creates a pixmap from the specified medium with a specified size.

*Signature*  make-pixmap medium width height => pixmap

**Parameters**

- **medium** – An instance of type <medium>.
- **width** – An instance of type <integer>. 
• **height** – An instance of type `<integer>\`.

**Values**

• **pixmap** – An instance of type `<pixmap>\`.

**Discussion** Creates a pixmap from medium with a specified size, in pixels, given by width and height.

**See also**

• `draw-pixmap`
  • `<pixmap>`
  • `pixmap?`

**move-to** Generic function

Move the position in the current path on the specified drawable.

**Signature** `move-to drawable x y => ()`

**Signature** `move-to* drawable point => ()`

**Parameters**

• **drawable** – An instance of type `type-union(<sheet>, <medium>)`.

  The following arguments are specific to `move-to`.

**Parameters**

• **x** – An instance of type `<real>`.
  • **y** – An instance of type `<real>`.

  The following argument is specific to `move-to*`.

**Parameters**

• **point** – An instance of type `<transform>`.

**Discussion**

Move the position in the current path on `drawable` to (x, y).

This function is used, in combination with `line-to, curve-to, and arc-to`, to define a path. The function `start-path` should be used to start the definition of the path, and `end-path` 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`
  • `curve-to`
  • `line-to`

**<pixmap>** Open Abstract Instantiable Class

The class of pixmap objects.

**Superclasses** `<image>`
Discussion

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 DUIM-DCs module.

- image-height
- image-width

See also

- draw-pixmap
- make-pixmap
- pixmap?

pixmap? Generic function

Returns true if the specified object is a pixmap.

Signature  pixmap? object => pixmap?

Parameters

- object  – An instance of type <object>.

Values

- pixmap?  – An instance of type <boolean>.

Discussion  Returns true if object is a pixmap.

See also

- <pixmap>

<pixmap-medium> Open Abstract Instantiable Class

The class of pixmap mediums.

Superclasses  <medium>

Discussion  The class of pixmap mediums, that is mediums capable of doing output to a pixmap.

Operations

- with-output-to-pixmap
See also

• <medium>
• with-output-to-pixmap

restore-clipping-region Generic function

start-path Generic function

Starts a new path on the specified drawable object.

Signature  start-path drawable => ()

Parameters

• drawable – An instance of type type-union(<sheet>, <medium>).

Discussion

Starts a new path on drawable. The path can be created with any number of calls to line-to, curve-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.

See also

• abort-path
• close-path
• end-path

stroke-path Generic function

Uses the current pen to draw the current path on the specified drawable object.

Signature  stroke-path drawable => ()

Parameters

• drawable – An instance of type type-union(<sheet>, <medium>).

Discussion  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.

See also

• close-path
• fill-path

with-output-to-pixmap Macro

Executes a body of code, returning the results to a pixmap.

Macro Call  with-output-to-pixmap (medium, #rest options) body end => pixmap

Parameters

• medium – An instance of type <pixmap-medium>.
• options – An instance of type <object>.
• body – An instance of type <object>.

Values

• pixmap – An instance of type <pixmap>.
Discussion

Executes a body of code, returning the results to a pixmap. Binds `medium` to a pixmap medium, that is, a medium that does output to a pixmap, and then evaluates `body` in that context. All the output done to `medium` inside of `body` is drawn on the pixmap stream. The pixmap medium supports the medium output protocol, including all of the graphics functions.

The returned value is a pixmap that can be drawn onto `medium` using `copy-from-pixmap`.

See also

- `do-with-output-to-pixmap`
- `<pixmap-medium>`
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. `DUIM-Layouts Module` 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.

The class hierarchy for DUIM-Layouts

This section presents an overview of the available classes of layout, and describes the class hierarchy present.

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 The class hierarchy for DUIM-Layouts. Only `<basic-composite-pane>`, `<leaf-pane>`, and `<layout>` have any subclasses defined. See Subclasses of `<layout>` for details of the subclasses of `<layout>`.

<table>
<thead>
<tr>
<th><code>&lt;sheet&gt;</code></th>
<th><code>&lt;basic-composite-pane&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;single-child-composite-pane&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;multiple-child-composite-pane&gt;</code></td>
</tr>
<tr>
<td><code>&lt;layout&gt;</code></td>
<td>See Subclasses of <code>&lt;layout&gt;</code></td>
</tr>
<tr>
<td><code>&lt;leaf-pane&gt;</code></td>
<td><code>&lt;null-pane&gt;</code></td>
</tr>
<tr>
<td><code>&lt;drawing-pane&gt;</code></td>
<td><code>&lt;simple-pane&gt;</code></td>
</tr>
<tr>
<td><code>&lt;simple-pane&gt;</code></td>
<td><code>&lt;top-level-sheet&gt;</code></td>
</tr>
</tbody>
</table>
All the actual layouts provided by the DUIM-Layouts library are subclasses of the base `<layout>` class, and are described in `Subclasses of `<layout>``. 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 DUIM-Geometry Library, and 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.

### Subclasses of `<layout>`

The subclasses of `<layout>` are shown in `Subclasses of the `<layout>` class`

<table>
<thead>
<tr>
<th><code>&lt;layout&gt;</code></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;row-layout&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;column-layout&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;fixed-layout&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;pinboard-layout&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;stack-layout&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;table-layout&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

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 8.1: Column, row, and pinboard layouts](image)

**DUIM-Layouts Module**

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

**allocate-space** Open Generic function

Allocates space within a layout for its children.

**Signature**

`allocate-space pane width height => ()`

**Parameters**

- `pane` – An instance of type `<sheet>`.
- `width` – An instance of type `<integer>`.
- `height` – An instance of type `<integer>`.

**Discussion**

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 `allocate-space` on each of the child panes. For example, `<column-layout>` arranges all its children 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`

**<basic-user-pane>** Class

The class of basic user panes.

**Superclasses** `<wrapping-layout-pane>`

**Init-Keywords**

• 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.


Discussion

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.

See also

• `define pane`

`<column-layout>` Open Abstract Instantiable Class

The class of column layouts.

Superclasses `<layout>`

Init-Keywords

• border – An instance of type `<integer>`. Default value: 0.

• spacing – An instance of type `<integer>`. Default value: 0.

• 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 – An instance of type `false-or(limited(<sequence>), of: <integer>))`. Default value: #f.

• y-ratios – An instance of type `false-or(limited(<sequence>), of: <integer>))`. Default value: #f.
Discussion

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.

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-border` 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.

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>
• <layout>
• <row-layout>
• <stack-layout>
• <table-layout>
• vertically

compose-space Generic function
Returns the amount of space required for a specified child of a composite pane.

Signature compose-space pane #key width height => space-req

Parameters
• pane – An instance of type <sheet>.
• width – An instance of type <integer>.
• height – An instance of type <integer>.

Values
• space-req – An instance of type <space-requirement>.

Discussion
Returns the amount of space required for pane, 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 compose-space. They answer by returning instances of <space-requirement>. 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.

The value returned by compose-space is an instance of <space-requirement> that represents how much space pane requires.

The width and height arguments are real numbers that the compose-space 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.

This function actually calls do-compose-space to perform the space calculations. Client code may specialize do-compose-space but should not call it. Call compose-space instead.

See also
• do-compose-space
• <space-requirement>
current-pane Generic function
Returns the current pane.

Signature  current-pane => pane

Parameters
• pane – An instance of type <sheet>.

Discussion  Returns the current pane: that is, the pane that has the mouse focus.

define pane Defining Macro
Defines a new class of DUIM pane.

Macro Call  define pane name ({$supers },*) {$slots-and-panes } end

Parameters
• name – A Dylan name bnf.
• supers – A Dylan name bnf.
• slots-and-panes – A Dylan body bnf.

Discussion  This macro lets you define a new class of DUIM pane.

The name argument represents the name of the new class of pane, and supers is a list of zero or more superclasses for the new class. Multiple superclass names are separated by commas.

The slots-and-panes argument represents the slot information for the new class, together with any init-keywords and default values that the slots should take.

Panes are sheets which represent a “useful unit” in a GUI. There is no protocol class called <pane>.

1. 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>.

2. Sometimes, a pane class implements some complex output-only sheet.

3. Sometimes, a pane class implements the <sheet> part of a <gadgets.htm#34543>.

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

do-allocate-space Open Generic function
Called by allocate-space to calculate space requirements for a pane.

Signature  do-allocate-space pane width height => ()

Parameters
• pane – An instance of type <sheet>.
• width – An instance of type <integer>.
• height – An instance of type <integer>.

Discussion This function is called by allocate-space 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 allocate-space instead.

See also
• allocate-space

do-compose-space Open Generic function
Called by compose-space to calculate space requirements for a child.

Signature do-compose-space pane #key width height => space-req

Parameters
• pane – An instance of type <sheet>.
• width – An instance of type <integer>.
• height – An instance of type <integer>.

Values
• space-req – An instance of type <space-requirement>.

Discussion This function is called by compose-space 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 do-compose-space explicitly: call compose-space instead.

Example Assume that you have defined a new class of scroll bar as follows:

```lisp
define class <my-scroll-bar> (<scroll-bar>, <leaf-pane>)
end class <test-scroll-bar>;
```

A new method for do-compose-space can be defined as follows:

```lisp
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

<drawing-pane> Open Abstract Instantiable Class
The class of drawing panes.

Superclasses <layout>

Init-Keywords

• display-function – An instance of type false-or(<function>). Default value: #f.

Discussion

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>.

See also

• handle-repaint
• pane-display-function
• <simple-pane>

$fill Constant
Default value for width and height init-keywords for layout panes.

Type <integer>
Value 100000

Discussion

This constant is used as the default value for any width: and height: init-keywords in layout panes.

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 min-width: or min-height: init-keywords are specified explicitly.

See also

• make

<fixed-layout> Open Abstract Instantiable Class
The class of fixed layouts.

Superclasses <layout>

Discussion

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.

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.
Fixed layouts are most useful if you know exactly what size and position every child in the layout should be.

See also

• <layout>
• <pinboard-layout>

<grid-layout> Open Abstract Instantiable Class
The class of grid layouts.

Superclasses <table-layout>

Init-Keywords

• cell-space-requirement – An instance of type <space-requirement>.

Discussion
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 cell-space-requirement: init-keyword lets you specify the preferred space requirement for any individual cell in the grid layout.

See also

• <column-layout>
• <row-layout>
• <stack-layout>
• <table-layout>

horizontally Statement Macro
Lays out a series of gadgets horizontally.

Macro Call horizontally ([options ]) {panes }+ end

Parameters

• options – Dylan arguments bnf.
• panes – One or more occurrences of Dylan body bnf.

Discussion
This macro lays a series of gadgets out horizontally, creating the necessary layouts for you automatically.

The options are passed directly to the row layout, and thus can be any legitimate combinations of init-keywords for <row-layout>. If no options are specified, then the default values for row layout are used.

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
<layout> Open Abstract Class

The superclass class of all layout classes.

Superclasses <sheet>

Init-Keywords

- **space-requrement** – 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: $fill$.
- **resizable?** – An instance of type `<boolean>`. Default value: #t.
- **fixed-width?** – An instance of type `<boolean>`. Default value: #f.
- **fixed-height?** – An instance of type `<boolean>`. Default value: #f.

Discussion

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.

The `space-requirement:` 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.

The `width:`. `height:`. `min-width:`. `min-height:`. `max-width:`. and `max-height:` 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.

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 resizeable using the `resizable?:` init-keyword. If `fixed-width?:` or `fixed-height?:` are true, then the layout cannot be resized in the appropriate direction. Setting both to #t is equivalent to setting `resizable?:` to #f. 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.

See also

- `<column-layout>`
- `<grid-layout>`
- `<pinboard-layout>`
- `<row-layout>`
- `<stack-layout>`
• \texttt{<table-layout>}

\textbf{layout-border} \textbf{Generic function}

Returns the amount of whitespace around the children in a layout.

\textbf{Signature} \quad \text{layout-border} \quad \text{layout} \Rightarrow \text{border}

\textbf{Parameters}

\begin{itemize}
  \item \textbf{layout} \quad \text{- An instance of type \texttt{type-union(<row-layout>,}
                  \texttt{<column-layout>, <table-layout>, <grid-layout>,}
                  \texttt{<stack-layout>).}}
\end{itemize}

\textbf{Values}

\begin{itemize}
  \item \textbf{border} \quad \text{- An instance of type \texttt{<integer>}.}
\end{itemize}

\textbf{Discussion}

Returns the amount of whitespace, in pixels, around the children in \texttt{layout}.

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.

\textbf{See also}

• \texttt{layout-border-setter}

\textbf{layout-border-setter} \textbf{Generic function}

Sets the amount of whitespace around the children in a layout.

\textbf{Signature} \quad \text{layout-border} \quad \text{border layout} \Rightarrow \text{border}

\textbf{Parameters}

\begin{itemize}
  \item \textbf{border} \quad \text{- An instance of type \texttt{<integer>}.}
  \item \textbf{layout} \quad \text{- An instance of type \texttt{type-union(<row-layout>,}
                   \texttt{<column-layout>, <table-layout>, <grid-layout>,}
                   \texttt{<stack-layout>).}}
\end{itemize}

\textbf{Values}

\begin{itemize}
  \item \textbf{border} \quad \text{- An instance of type \texttt{<integer>}.}
\end{itemize}

\textbf{Discussion}

Sets the amount of whitespace, in pixels, around the children in \texttt{layout}.

You can also set this value a layout is created using the \texttt{border: init-keyword}.

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.

\textbf{See also}

• \texttt{layout-border}

\textbf{layout-equalize-heights?} \textbf{Generic function}

Returns true if the children of the specified layout are all the same height.

\textbf{Signature} \quad \text{layout-equalize-heights?} \quad \text{layout} \Rightarrow \text{equal?}

\textbf{Parameters}

\begin{itemize}
  \item \textbf{layout} \quad \text{- An instance of type \texttt{type-union(<row-layout>,}
                       \texttt{<column-layout>)}.
\end{itemize}
Values

• **equal?** – An instance of type `<boolean>`.

Discussion

Returns true if the children of `layout` 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-heights?: init-keyword`. There is no equivalent setter function.

See also

• `layout-equalize-widths?`

layout-equalize-widths? Generic function

Returns true if the children of the specified layout are all the same width.

**Signature** layout-equalize-widths? layout => equal?

**Parameters**

• **layout** – An instance of type type-union(<row-layout>, <column-layout>)

**Values**

• **equal?** – An instance of type `<boolean>`.

Discussion

Returns true if the children of `layout` are all the same width. 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-heights?`

<leaf-pane> Open Abstract Class

The class of leaf panes.

**Superclasses** `<sheet>`

**Discussion**

The class of leaf panes. These are sheets that live at the leaf of the sheet tree that obeys the layout protocols.

Subclass this class if you want to create a basic leaf pane.

• If you want to do output to it, mix in one of the `<sheet-with-medium-mixin>` classes.

• If you want to do input from it, min in one of the `<sheet-with-event-queue>` classes.

• If you want to repaint it, mix in one of the `<sheet-with-repainting-mixin>` classes.

make(<space-requirement>) Method

Creates an instance of `<space-requirement>`.

**Signature** make space-requirement-class #key width min-width max-width height min-height max-height => space-req
Parameters

- `space-requirement-class` – The class `<space-requirement>`.
- `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`.
- `min-height` – An instance of type `<integer>`. Default value: `height`.

Values

- `space-req` – An instance of type `<space-requirement>`.

Discussion

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>`, for a full description of this behavior.

See also

- `$fill`
- `<space-requirement>`

<multiple-child-composite-pane> Open Abstract Class

The class of composite panes that can have multiple children.

Superclasses `<layout>`

Discussion 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.

See also

- `<single-child-composite-pane>`

<null-pane> Instantiable Sealed Class

The class of null panes.

Superclasses `<leaf-pane>`

Discussion The class of null panes. This class acts as a filler: use it when you need to “fill space” somewhere in a complex layout.

See also

- `<spacing>`
- `with-spacing`

pane-display-function Generic function

Returns the function used to display the specified pane.

Signature `pane-display-function pane => pane-display-function`

Parameters

- `pane` – An instance of type `<sheet>`.
Keyword pane-display-function  An instance of type false-or(<function>).

Discussion

Returns the function used to display pane, where pane is any pane that can have a
display-function: init-keyword specified. The value returned by pane-display-function
is the value of the display-function: init-keyword.

The display function gets called by the handle-repaint method for <simple-pane> and
<drawing-pane>.

See also
* <drawing-pane>

pane-layout Generic function

Returns the layout that contains the specified pane in define pane.

Signature  pane-layout pane => layout-pane

Parameters

* pane – An instance of type <sheet>.

Values

* layout-pane – An instance of type <sheet>.

Discussion  Returns the layout that contains the specified pane in define pane.

See also
* define pane

<pinboard-layout> Open Abstract Instantiable Class

The class of pinboard layouts.

Superclasses  <layout>

Init-Keywords

* stretchable? – An instance of type <boolean>.

Discussion

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.

![Pinboard Layout Diagram](image)

Fig. 8.2: Three buttons arranged in a pinboard layout

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 been speci-
ified 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).

See also
- `<fixed-layout>`
- `<layout>`

**relayout-children**

Generic function

Lays out the children of the specified sheet again.

**Signature**

```
relayout-children sheet #key port-did-it? => ()
```

**Parameters**

- `sheet` – An instance of type `<sheet>`.

**Discussion**

Lays out the children of `sheet` again.

See also
- `relayout-parent`

**relayout-parent**

Generic function

Lays out the parent of the specified sheet again.

**Signature**

```
relayout-parent sheet #key width height => ()
```

**Parameters**

- `sheet` – An instance of type `<sheet>`.
- `width` – An instance of type `<integer>`.
- `height` – An instance of type `<integer>`.

**Discussion**

Lays out the parent of `sheet` again. If `width` and `height` are specified, then the parent is laid out in accordance with these dimensions.

See also
- `relayout-children`

**<row-layout>**

Open Abstract Instantiable Class

The class of row layouts.

**Superclasses** `<layout>`

**Init-Keywords**

- `border` – An instance of type `<integer>`. Default value: 0.
- `x-spacing` – An instance of type `<integer>`. Default value: 0.
- `spacing` – An instance of type `<integer>`. Default value: 0.
• **x-ratios** – An instance of type `false-or(<sequence>)`. Default value: `#f`.

• **ratios** – An instance of type `false-or(<sequence>)`. Default value: `#f`.

**Discussion**

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.

![Diagram of three buttons arranged in a row layout](image)

**Fig. 8.3:** 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-border` 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.

![Diagram of child allocation](image)

**Example**  To make a row of buttons that are all the same size:
See also

- `<column-layout>`
- `horizontally`
- `<layout>`
- `<grid-layout>`
- `<stack-layout>`
- `<table-layout>`

### `<simple-pane>` Open Abstract Instantiable Class
The class of simple panes.

**Superclasses** `<layout>`

**Init-Keywords**

- `display-function` – An instance of type `false-or(<function>)`. Default value: `#f`.

**Discussion**

The class of simple panes.

The `display-function`: init-keyword defines the display function for the pane. This gets called by the `handle-repaint` method for `<simple-pane>`.

See also

- `<drawing-pane>`
- `handle-repaint`
- `pane-display-function`

### `<single-child-composite-pane>` Open Abstract Class
The class of composite panes that can only have one child.

**Superclasses** `<layout>`

**Discussion** The class of composite panes that can only have one child.

See also

- `<multiple-child-composite-pane>`

### `<space-requirement>` Abstract Instantiable Class
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`.
• **min-height** – An instance of type `<integer>`. Default value: `height`.
• **max-height** – An instance of type `<integer>`. Default value: `height`.
• **label** – An instance of type `type-union(<string>, <image>)`.

**Discussion**

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.

The various init-keywords let you constrain the width and height of the object in a variety of ways.

If no init-keywords are specified, the object returned tries to fill all the available space.

Specifying `width:` or `height:` specifies the preferred width or height of the object.

Specifying any of the `min-` or `max-` init-keywords lets you minimum and maximum width or height for the object.

The following inequalities hold for all widths and heights:

\[
\text{min-height:} \leq \text{height:} \leq \text{max-height:} \quad \text{min-width:} \leq \text{width:} \leq \text{max-width:}
\]

If either `min-width:` or `min-height:` is 0, the object is “infinitely shrinkable” in that direction. If either `max-width:` or `max-height:` is `$fill`, 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.

An example of the use of `max-width:` to force the size of a button to fit the available space can be found in the entry for `<button>`.

The `label:` init-keyword specifies a label which is measured to give the preferred width and height.

**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:

```duim
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.

```duim
define method do-compose-space
  (pane :: <basic-test-button>, #key width, height)
=> (space-req :: <space-requirement>)
  ignore(width, height);
  make(<space-requirement>),
```

8.3. DUIM-Layouts Module
See also

- \$fill

**space-requirement?** Generic function
Returns true if the specified object is a space requirement.

**Signature**

```
space-requirement? object => boolean
```

**Parameters**

- **object** – An instance of type `<object>`.

**Values**

- **boolean** – An instance of type `<boolean>`.

**Discussion** Returns true if `object` is an instance of `<space-requirement>`.

See also

- `<space-requirement>`

**space-requirement-height** Generic function
Returns the preferred height of the specified space requirement.

**Signature**

```
space-requirement-height sheet space-req => height
```

**Parameters**

- **sheet** – An instance of type `<sheet>`.
- **space-req** – An instance of type `<space-requirement>`.

**Values**

- **height** – An instance of type `<number>`.

**Discussion** Returns preferred the height of `space-req`. This is the value of the `height:` init-keyword that was passed when the object was created.

See also

- `space-requirement-max-height`
- `space-requirement-min-height`

**space-requirement-max-height** Generic function
Returns the maximum allowed height of the specified space requirement.

**Signature**

```
space-requirement-max-height sheet space-req => max-height
```

**Parameters**

- **sheet** – An instance of type `<sheet>`.
- **space-req** – An instance of type `<space-requirement>`.

**Values**

- **max-height** – An instance of type `<number>`.

**Discussion** Returns the maximum allowed height of `space-req`. This is the value of the `max-height:` init-keyword that was passed when the object was created.
See also

- `space-requirement-height`
- `space-requirement-min-height`

**space-requirement-max-width**

Generic function

Returns the maximum allowed width of the specified space requirement.

**Signature**

`space-requirement-max-width sheet space-req => max-width`

**Parameters**

- `sheet` – An instance of type `<sheet>`.
- `space-req` – An instance of type `<space-requirement>`.

**Values**

- `max-width` – An instance of type `<number>`.

**Discussion**

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`
- `space-requirement-width`

**space-requirement-min-height**

Generic function

Returns the minimum allowed height of the specified space requirement.

**Signature**

`space-requirement-min-height sheet space-req => min-height`

**Parameters**

- `sheet` – An instance of type `<sheet>`.
- `space-req` – An instance of type `<space-requirement>`.

**Values**

- `min-height` – An instance of type `<number>`.

**Discussion**

Returns the minimum allowed height of `space-req`. This is the value of the `min-height`: init-keyword that was passed when the object was created.

See also

- `space-requirement-height`
- `space-requirement-max-height`

**space-requirement-min-width**

Generic function

Returns the minimum allowed width of the specified space requirement.

**Signature**

`space-requirement-min-width sheet space-req => min-width`

**Parameters**

- `sheet` – An instance of type `<sheet>`.
- `space-req` – An instance of type `<space-requirement>`.

**Values**

- `min-width` – An instance of type `<number>`. 
Discussion  Returns the minimum allowed width of space-req. This is the value of the min-width: init-keyword that was passed when the object was created.

See also

• space-requirement-max-width

• space-requirement-width

space-requirement-width Generic function
Returns the preferred width of the specified space requirement.

Signature  space-requirement-width sheet space-req => width

Parameters

• sheet – An instance of type <sheet>.

• space-req – An instance of type <space-requirement>.

Values

• width – An instance of type <number>.

Discussion  Returns the preferred width of space-req. This is the value of the width: init-keyword that was passed when the object was created.

See also

• space-requirement-max-width

• space-requirement-min-width

<stack-layout> Open Abstract Instantiable Class
The class of stack layouts.

Superclasses  <layout>

Init-Keywords

• border – An instance of type <integer>. Default value: 0.

• mapped-page – An instance of <sheet>.

Discussion

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-border 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.

See also

• <column-layout>

• <grid-layout>
• `<layout>`
• `<row-layout>`
• `<table-layout>`

**stack-layout-mapped-page Generic function**
Returns the currently mapped page for a stack layout.

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

**Parameters**
- `stack-layout` – An instance of `<stack-layout>`.

**Values**
- `page` – An instance of `<sheet>`.

**Discussion**
Returns the currently mapped `page` for the specified `stack-layout`.

**stack-layout-mapped-page-setter**
Generic function
Sets the mapped page for a stack layout.

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

**Parameters**
- `page` – An instance of `<sheet>`.
- `stack-layout` – An instance of `<stack-layout>`.

**Values**
- `page` – An instance of `<sheet>`.

**Discussion**
Sets the mapped page for the specified `stack-layout` to `page`.

**table-contents**
Generic function
Returns the contents of the specified table.

**Signature**
`table-contents table => contents`

**Parameters**
- `table` – An instance of type `<table-layout>`.

**Values**
- `contents` – An instance of type `<sheet>`.

**Discussion**
Returns the contents of `table`.
See also
- `table-contents-setter`

**table-contents-setter**
Generic function
Sets the contents of the specified table.

**Signature**
`table-contents-setter contents table => contents`

**Parameters**
- `contents` – An instance of type `<sheet>`.
- `table` – An instance of type `<table-layout>`.

**Values**
• **contents** – An instance of type `<sheet>`.

**Discussion** Sets the contents of `table`.

**See also**

• `table-contents`

**<table-layout> Open Abstract Instantiable Class**

The class of table layouts.

**Superclasses** `<layout>`

**Init-Keywords**

• **border** – An instance of type `<integer>`. Default value: 0.

• **rows** – An instance of type `false-or(<integer>)`. Default value: `#f`.

• **columns** – An instance of type `false-or(<integer>)`. Default value: `#f`.

• **contents** – An instance of type `limited(<sequence>, of: limited(<sequence>, of: <sheet>))`.

• **x-spacing** – An instance of type `<integer>`. Default value: 0.

• **y-spacing** – An instance of type `<integer>`. Default value: 0.

• **x-ratios** – An instance of type `false-or(<sequence>)`. Default value: `#f`.

• **y-ratios** – An instance of type `false-or(<sequence>)`. 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"`.

**Discussion**

The class of table layouts.

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-border` around the layout, except it uses a simpler syntax.

The **rows**: and **columns**: init-keywords are used to specify the number of rows and columns for the table layout.

The **contents**: 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 `contents` 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.

The **x-spacing**: and **y-spacing**: init-keywords let you specify how much vertical and horizontal space should be inserted, in pixels, between the children of the layout.

The **x-ratios**: and **y-ratios**: init-keywords let you specify the proportion of the total horizontal and vertical space that should be taken up by each individual child.

The value passed to **x-ratios**: needs to be a sequence of as many integers as there are columns of children in the layout. The value passed to **y-ratios**: 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.
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 \texttt{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.

![Diagram of x-ratios: \( (1, 2, 3) \)]

Alternatively, if the value \( (1, 2, 3) \) is specified for the \texttt{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.

![Diagram of y-ratios: \( (1, 2, 3) \)]

Finally, if both the \texttt{x-ratios:} and \texttt{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 \texttt{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 \texttt{y-alignment:} keyword.

**Operations**

- \texttt{table-contents}
- \texttt{table-contents-setter}

**Example**

```plaintext
*t* := make(<vector>, size: 9);
for {i from 1 to 9}
```
x-ratios: #(1, 2, 3)

\[
\begin{array}{|c|c|c|}
\hline
\text{\texttt{\textdagger\textendash}} & \text{\textdagger} & \text{\textdagger}\textendash \\
\hline
1 & 2 & 3 \\
\hline
\end{array}
\]

y-ratios: #(1, 2, 3)

\[
\begin{array}{|c|c|c|}
\hline
\text{\texttt{\textdagger\textendash}} & \text{\textdagger} & \text{\textdagger}\textendash \\
\hline
1 & 2 & 3 \\
\hline
\end{array}
\]

\[
\texttt{\textdagger\textendash}[i - 1] := \text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}}} \\
\texttt{\textdagger\textendash}\text{\texttt{\textdagger\textendash}}; \\
\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}}} \\
\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}}} \\
\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}}} \\
\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}}} \\
\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}}} \\
\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}}} \\
\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}(\text{\texttt{\textdagger\textendash}}, \text{\texttt{\textdagger}})
\]

See also
- <column-layout>
- <grid-layout>
- <layout>
- <row-layout>
- <stack-layout>
- tabling

**tablging Statement Macro**
Lays out a series of gadgets in a table.

**Macro Call**

```
tablging (options ) {panes }+ end
```

**Parameters**

- **options** – Dylan arguments *bnf*.
- **panes** – One or more occurrences of Dylan body *bnf*.

**Discussion**

This macro lays a series of gadgets out in a table, creating the necessary layouts for you automatically.

The *options* are passed directly to the table layout, and thus can be any legitimate combinations of init-keywords for `<table-layout>`. If no options are specified, then the default values for table layout are used.

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
- `<table-layout>`
- vertically

`<top-level-sheet>` Open Abstract Instantiable Class

The class of top level sheets.

**Superclasses**  `<layout>`

**Init-Keywords**

- **display** – An instance of type `false-or(<display>)`. Default value: `#f`.
- **frame** – An instance of type `false-or(<frame>)`. Default value: `#f`.
- **frame-manager** – An instance of type `false-or(<frame-manager>)`. Default value: `#f`.
- **container** – An instance of type `false-or(<object>)`. Default value: `#f`.
- **container-region** – An instance of type `false-or(<region>)`. Default value: `#f`.

**Discussion**

The class of top level sheets.

The `container:` and `container-region:` init-keywords are for use in embedded frames, such as OLE objects in HTML browser windows. The `container:` init-keyword denotes the container itself, and `container-region:` 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.

`vertically` Statement Macro

Lays out a series of gadgets vertically.

**Macro Call**

`vertically ((options)) {panes}+ end`

**Parameters**

- **options** – Dylan arguments `bnf`.
- **panes** – One or more occurrences of Dylan body `bnf`.

**Discussion**

This macro lays a series of gadgets out vertically, creating the necessary column layout for you automatically.

The `options` are passed directly to the column layout, and thus can be any legitimate combinations of init-keywords for `<column-layout>`. If no options are specified, then the default values for table layout are used.

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.

**Example**

```dylan
contain(vertically (border: 5, equalize-widths: #t)
make(<button>, label: "Hello")
make(<button>, label: "World")
end);
```
See also

- `<column-layout>`
- `horizontally`
- `tabling`
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. *DUIM-Gadgets Module* 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 *DUIM-Gadgets Module*. For a more general introduc-
tion 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.

**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.

**Gadget protocols**

Gadgets are objects that make up an interface: the menus, buttons, sliders, check lists, tool bars, menu bars, and so on. Gadgets 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.

**gadget-label** This slot holds the label that appears on the gadget on the screen. If a gadget does not have a label, the gadget-label function returns #f.

**gadget-value** This slot holds the value(s) of the gadget. If a gadget does not have any values, the gadget-value function returns #f.
This slot is a list of the contents of the gadget. If the gadget does not have items, for example a button, \texttt{gadget-items} returns nothing.

\textbf{gadget-enabled?} This slot tests whether or not the gadget is active. All gadgets have a \texttt{gadget-enabled?} slot.

An introduction to the protocols supported by different sorts of gadget can also be found in the \textit{Building Applications using DUIM} book.

\section*{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 \textit{items} protocol are displayed in \textit{bold text}, and classes that support the activate protocol are displayed using \textit{italic text}.

\textit{Note:} In \textit{Subclasses of the} \texttt{<collection-gadget>} \textit{class}, every subclass shown supports the \textit{items} protocol, though for clarity, no bold is used.

All subclasses of \texttt{<value-gadget>} support the \textit{value} protocol. These are described in \textit{Subclasses of <value-gadget>}, \textit{Subclasses of <button>}, and \textit{Subclasses of <collection-gadget>}

\section*{The \texttt{<gadget>} class and its subclasses}

The base class for the majority of DUIM gadgets is the \texttt{<gadget>} class, which is itself a subclass of \texttt{<object>}. All other DUIM gadgets are subclasses of \texttt{<gadget>}, with the exception of \texttt{<list-item>, <tree-node>}, and \texttt{<table-item>}

The immediate subclasses of \texttt{<gadget>} are shown in \textit{Overall class hierarchy for the DUIM-Gadgets library}. Only \texttt{<value-gadget>} and \texttt{<page>} have any subclasses defined. See \textit{Subclasses of <value-gadget>} and \textit{Subclasses of <page>} for details of these subclasses.

The \texttt{<gadget>} class provides a number of subclasses that allow particular parts of a user interface to be created:

\texttt{<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 \textit{Subclasses of <button>} and \textit{Subclasses of <collection-gadget>}

\texttt{<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.

\texttt{<scroller>} This is a generic scrolling gadget that can be used in a number of situations.

\texttt{<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.

\texttt{<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 \texttt{<gadget>} class provides a number of subclasses that allow general spatial and grouping capability, in addition to the layout functionality described in DUIM-Layouts Library. These are as follows:

\texttt{<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.

\texttt{<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.
This allows you to specify how much space should be placed between any two gadgets or groups of gadgets.

This allows a visible border to be placed around any number of gadgets.

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.

Overall class hierarchy for the DUIM-Gadgets library

See 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 Overview.

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 Subclasses of <button> 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 Subclasses of <collection-gadget> 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 ‘Subclasses of the <value-gadget> class’.

Subclasses of the <value-gadget> class
<value-gadget>
<text-gadget>
<password-field>
<text-editor>
<text-field>
<value-range-gadget>
<slider>
<scroll-bar>
<progress-bar>
<button>
See Subclasses of <button>
<menu-bar>
<status-bar>
<tab-control>
<collection-gadget>
See Subclasses of <collection-gadget>

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 ‘Subclasses of the <page> class’.

Subclasses of the <page> class
<page>
<tab-control-page>
<property-page>
The `<tab-control-page>` class is used to define the elements that are associated with each tab in a tab control.

![Fig. 9.1: A tab control page](image)

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.

![Fig. 9.2: A property page](image)

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.

![Fig. 9.3: A wizard page](image)

### Subclasses of `<button>`

The subclasses of `<button>` are as shown in ‘Subclasses of the `<button>` class’. 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 `Overview`.

Subclasses of the `<button>` class

- `<button>`
- `<check-button>`
Subclasses of `<collection-gadget>`

The subclasses of `<collection-gadget>` are as shown in ‘Subclasses of the `<collection-gadget>` class’. All of these subclasses support the `items` protocol, even though they are not displayed in bold.

Subclasses of the `<collection-gadget>` class

- `<button-box>`
- `<check-box>`
- `<push-box>`
- `<radio-box>`
- `<list-box>`
- `<menu-box>`
- `<check-menu-box>`
- `<push-menu-box>`
- `<radio-menu-box>`
- `<option-box>`
- `<combo-box>`
- `<spin-box>`
- `<list-control>`
- `<tree-control>`
- `<table-control>`

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 Overview.

### 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.

![Fig. 9.4: A selection of button and equivalent menu buttons](image)

### 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.

<text-editor>

<password-field>
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.

![Fig. 9.5: A variety of simple collection gadgets](image)

Value range gadgets

A value range gadget is any gadget whose value falls within a defined Dylan range.

![Fig. 9.6: A variety of value range gadgets](image)

Sliders, scroll bars, and progress 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 ‘A variety of value range gadgets’. 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.
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 ‘<frames.htm#89815>’.

<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 ‘<frames.htm#89815>’.

Note: The <wizard-page> and <property-page> classes are actually exposed by the DUIM-Frames library, rather than the DUIM-Gadgets library. See ‘<frames.htm#89815>’ for full details on this library.

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.

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.
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.

![Fig. 9.8: A status bar](image)

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.

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.

![Fig. 9.9: A tab control](image)

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 Page gadgets.

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.

DUIM-Gadgets Module

This section contains a complete reference of all the interfaces that are exported from the `duim-gadgets` module.
<action-gadget> Open Abstract Class

The protocol class for gadgets that have action callbacks.

Superclasses <gadget>

Init-Keywords

• activate-callback – An instance of type false-or(<function>). Default value: #f.

Discussion 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

• gadget-activate-callback
• gadget-activate-callback-setter

See also

• <gadget>

activate-gadget Generic function

Activates the specified gadget.

Signature activate-gadget gadget => ()

Parameters

• gadget – An instance of type <gadget>.

Discussion 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.

add-column Generic function

Adds a column to the specified table.

Signature add-column table heading generator index => ()

Parameters

• table – An instance of type <table-control>.
• heading – An instance of type type-union(<string>, <label>).
• generator – An instance of type <function>.
• index – An instance of type <integer>.

Discussion Adds a column table, with a table heading given by heading. The contents of the column
are generated by calling the generator function on the item for each row of table. The index
specifies where in the column order the new column should be added.

See also

• remove-column

add-item Generic function

Adds an item to the specified list or table control.

Signature add-item list-or-table item #key after => item

Parameters

• list-or-table – An instance of type-union(<list-control>, <table-control>).
• **item** – An instance of type `type-union(<list-item>, <table-item>)`.

• **after** (#key) – An instance of type `type-union(<list-item>, <table-item>)`.

**Values**

• **item** – An instance of type `type-union(<list-item>, <table-item>)`.

**Discussion**

Adds an *item* to the specified *list-or-table*. The new item is created via a call to `make-item`. The *after* argument indicates which item to place the new item after.

See also

• `find-item`

• `<list-control>`

• `<list-item>`

• `make-item`

• `remove-item`

• `<table-control>`

• `<table-item>`

---

**add-node** Generic function

Adds node to the specified tree control.

**Signature** `add-node tree parent node #key after setting-roots? => node`

**Parameters**

• **tree** – An instance of `<tree-control>`.

• **parent** – An instance of `<tree-control>`.

• **node** – An instance of type `<tree-node>`.

• **after** (#key) – An instance of type `<tree-node>`.

• **setting-roots?** (#key) – An instance of type `<boolean>`.

**Values**

• **node** – An instance of type `<tree-node>`.

**Discussion**

Adds a *node* to the specified *tree* with the specified *parent*. The new item is created via a call to `make-node`.

The *after* argument indicates which node to place the new node after. If `setting-roots?` is true, then the new node is added at the root of *tree*.

See also

• `find-node`

• `make-node`

• `remove-node`

• `<tree-control>`
<border> Open Abstract Instantiable Class
The class of bordering gadgets.

Superclasses <gadget> <single-child-composite-pane>

Init-Keywords
  • thickness – An instance of type <integer>. Default value: 1.
  • type – An instance of type one-of(#f, #"flat", #"sunken", #"raised", #"ridge", #"groove", #"input", #"output"). Default value: #f.

Discussion
The base class of gadgets that provide borders to their children.

The thickness of the border is specified by the thickness: init-keyword, and is given in pixels.

The type: 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.

Borders are usually created using the with-border macro, rather than by making direct instances of this class.

See also
  • <group-box>
  • with-border

<button> Open Abstract Instantiable Class
The class of all button gadgets.

Superclasses <value-gadget>

Init-Keywords
  • accelerator – An instance of type false-or(<gesture>). Default value: #f.
  • mnemonic – An instance of type false-or(<character>). Default value: #f.

Discussion
The class of all button gadgets.

The accelerator: 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**

- `<frames.htm#56017>`
- `<frames.htm#56015>`
- `<frames.htm#24406>`
- `<frames.htm#37806>`
- `<frames.htm#48310>`
- `<frames.htm#91817>`
- `<frames.htm#56017>`

**Example**

```make
(make <button>, label: "Hello",
   activate-callback:
   method (gadget)
   notify-user
   { format-to-string
     ("Pressed button %=", gadget),
     owner: gadget)
   end));
```

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.

```plaintext
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>`
- `<check-button>`
- `$fill`
- `gadget-accelerator`
- `<menu-button>`
- `<radio-button>`
- `<space-requirement>`

**<button-box>** Open Abstract Instantiable Class

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.

**Discussion**

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 \texttt{layout-class}: is used to parent the buttons that are created, and any extra arguments that are specified, such as \texttt{x-alignment:} and \texttt{x-spacing:}, are passed along to this layout.

You can use the \texttt{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 \texttt{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 \texttt{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 \texttt{child:} init-keyword has been used rather than the \texttt{items:} init-keyword.

If you use \texttt{child:}, then the \texttt{gadget-value} returned by the gadget is the \texttt{gadget-id} of the selected button. Contrast this with \texttt{items:}, where the selected item is returned as the \texttt{'gf:gadget-value'}.

Examples

\begin{verbatim}
contain(make(<button-box>,
  selection-mode: #"multiple",
  items: range(from: 0, to: 20)));
\end{verbatim}

The following examples illustrate the use of some of the init-keywords described. They each create an instance of a subclass of \texttt{<button-box>}. Note that the \texttt{selection-mode:} init-keyword may be used instead, rather than creating a direct instance of one of the subclasses.

\begin{verbatim}
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));
\end{verbatim}

See also

- \texttt{<check-box>}
- \texttt{<push-box>}
- \texttt{<radio-box>}

\texttt{<check-box>} Open Abstract Instantiable Class

The class of check boxes, or groups of check buttons.

Superclasses \texttt{<button-box> <action-gadget>}

Discussion

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.
Examples

```
contain(make(<check-box>, items: #(1, 2, 3, 4, 5)));
contain(make(<check-box>, items: range(from: 1, to: 9),
           columns: 3));
contain(make(<check-box>, items: #(1, 2, 3, 4),
           layout-class: <table-layout>
           rows: 2));
```

See also

- `<group-box>`
- `<push-box>`
- `<radio-box>`

**<check-button>** Open Abstract Instantiable Class

The class of check buttons.

**Superclasses** `<button>` `<action-gadget>`

**Discussion**

The class of check buttons. The value of a check button is either `#t` or `#f`, depending whether or not it is currently selected.

Internally, this class maps into the check box Windows control.

**Example**

```
contain(make(<check-button>, label: "Check button"));
```

See also

- `<check-menu-button>`
- `<push-button>`
- `<radio-button>`

**<check-menu-box>** Open Abstract Instantiable Class

The class of groups of check buttons displayed in a menu.

**Superclasses** `<menu-box>` `<action-gadget>`

**Discussion**

The class of groups of check buttons displayed in a menu.

Internally, this class maps into the menu Windows control.

**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")),
           )));
```
<check-menu-button> Open Abstract Instantiable Class

The class of check buttons that can be displayed in a menu.

Superclasses  <menu-button>

Discussion

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.

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>
• <radio-menu-button>

<collection-gadget> Open Abstract Class

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(<frames.htm#40934>, <function>).

Discussion

The class of all gadgets that can contain collections.
The **items**: init-keyword is used to specify the collection of items that the collection gadget contains.

The **label-key** and **value-key**: init-keywords are functions that are used to calculate the labels and the value of the gadget respectively.

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.

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.

By default, the value key returns the collection gadget itself.

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`, for a fuller description of key-press callbacks.

**Operations**

- `gadget-items`
- `gadget-items-setter`
- `gadget-key-press-callback`
- `gadget-key-press-callback-setter`
- `gadget-label-key`
- `gadget-selection`
gadget-selection-mode

• gadget-selection-setter

• gadget-test

• gadget-value-key

See also

• <button-box>

• <check-box>

• <check-menu-box>

• <combo-box>

• <list-box>

• <list-control>

• <option-box>

• <radio-box>

• <radio-menu-box>

• <table-control>

• <tree-control>

<combo-box> Open Abstract Instantiable Class

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".

Discussion

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 dropdown 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.

Example
contain(make(<combo-box>, value-type: <integer>,
    items: range(from: 1 to: 5)));
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

- `contract-node`
- `tree-control-children-generator`

**find-item** Generic function

Finds an item in a list control or a table control.

**Signature**

```lisp
find-item list-or-table object #key => found-item
```

**Parameters**

- `list-or-table` – An instance of `type-union(<list-control>, <table-control>)`.
- `object` – An instance of type `<object>`.

**Values**

- `found-item` – An instance of type `type-union(<list-item>, <table-item>, #f)`.

**Discussion**

Finds the item in a list control or a table control that corresponds to `object`.

See also

- `add-item`
- `<list-control>`
- `list-item`
- `make-item`
- `remove-item`
- `<table-control>`
- `<table-item>`

**find-node** Generic function

Finds a node in a tree control.

**Signature**

```lisp
find-item tree object #key parent-node => found-item
```

**Parameters**

- `tree` – An instance of `<tree-control>`.
- `object` – An instance of `<object>`.
- `parent-node (#key)` – An instance of type `<tree-node>`.

**Values**

- `found-item` – An instance of type `<tree-node>`.

**Discussion**

Finds the item in a tree control that corresponds to `object`.

See also

- `add-node`
- `make-node`
- `remove-node`
<gadget> Open Abstract Class

The protocol class of all gadgets.

Supertypes: <object>

Init-Keywords

- **id** – An instance of type false-or(<object>). Default value: #f.
- **client** – An instance of type false-or(<object>). Default value: #f.
- **label** – An instance of type type-union(<string>, <image>). Required.
- **documentation** – An instance of type false-or(<string>). Default value: #f.
- **enabled?** – An instance of type <boolean>. Default value: #t.
- **read-only?** – An instance of type <boolean>. Default value: #f.

Discussion

The class of all gadgets. You should not create a direct instance of this class.

The id: 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:

```lisp
select (gadget-id)
  "ok" => do-okay();
  "cancel" => do-cancel();
end select;
```

Note, however, that specifying id: is not generally necessary. The id: init-keyword is useful in the case of tab controls, where it is returned by gadget-value.

Every gadget has a client: that is specified when the gadget is created. Typically, client: is a frame or a composite sheet.

The label: 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).

The documentation: 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 toolbar may display a tooltip (a piece of pop-up text) that contains the supplied documentation.

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

- **activate-gadget**
• choose-from-dialog
• gadget-accelerator
• gadget-accelerator-setter
• gadget-client
• gadget-client-setter
• gadget-command
• gadget-command-setter
• gadget-default?
• gadget-default?-setter
• gadget-documentation
• gadget-documentation-setter
• gadget-value-changing-callback
• gadget-value-changing-callback-setter
• gadget-enabled?
• gadget-enabled?-setter
• gadget-id
• gadget-id-setter
• gadget-label
• gadget-label-setter
• gadget-mnemonic
• gadget-mnemonic-setter
• gadget-orientation
• gadget-popup-menu-callback
• gadget-popup-menu-callback-setter
• gadget-read-only?
• gadget-scrolling-horizontally?
• gadget-scrolling-vertically?
• update-gadget

See also
• <action-gadget>
• <border>
• gadget-value
• <group-box>
• <label>
• <menu>
• <page>
gadget?  Generic function
Returns true if the specified object is a gadget.

Signature  gadget? object => gadget?
Parameters
•  object – An instance of type <object>.

Values
•  gadget? – An instance of type <boolean>.

Discussion  Returns true if object is a gadget.

Example

```
gadget* := contain(make
  (<radio-menu-box>,
   items: range(from: 0, to: 20)));
gadget?(gadget*);
```

See also
•  <gadget>

gadget-accelerator  Generic function
Returns the keyboard accelerator of the specified gadget.

Signature  gadget-accelerator gadget => accelerator
Parameters
•  gadget – An instance of type <gadget>.

Values
•  accelerator – An instance of type <gesture>.

Discussion  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
•  <button>
•  gadget-accelerator-setter
•  <gesture>
•  <menu-button>
gadget-accelerator-setter Generic function
Sets the keyboard accelerator of the specified gadget.

Signature  gadget-accelerator-setter accelerator gadget => accelerator

Parameters
• accelerator – An instance of type <gesture>.
• gadget – An instance of type <gadget>.

Values
• accelerator – An instance of type <gesture>.

Discussion
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>
• gadget-accelerator
• <gesture>
• <menu-button>

gadget-activate-callback Generic function
Returns the activate callback of the specified gadget.

Signature  gadget-activate-callback gadget => activate-callback

Parameters
• gadget – An instance of type <action-gadget>.

Values
• activate-callback – An instance of type false-or(<function>).

Discussion
Returns the function that will be called when gadget is activated. This function will be invoked with one argument, the gadget itself.

When this function returns #f, this indicates that there is no activate callback for the gadget.

See also
• gadget-activate-callback-setter

gadget-activate-callback-setter Generic function
Sets the activate callback for the specified gadget.

Signature  gadget-activate-callback-setter activate-callback gadget => activate-callback

Parameters
• activate-callback – An instance of type false-or(<function>).
• gadget – An instance of type <action-gadget>.

Values
• activate-callback – An instance of type false-or(<function>).
Discussion  Sets the activate callback for gadget to activate-callback.

See also

• gadget-activate-callback

**gadget-client**  Generic function

Returns the client of the specified gadget.

**Signature**  gadget-client gadget => client

**Parameters**

- gadget – An instance of type <gadget>.
- client – An instance of type <object>.

**Discussion**

Returns the client of gadget. The client is the gadget or frame that gadget should look to for callback information.

In any sheet hierarchy, the client is usually the immediate parent of gadget. 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 gadget-id, you are free to assign your own client to a given gadget whenever it is necessary for your code.

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 <action-gadget>, 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-setter

**gadget-client-setter**  Generic function

Sets the client of the specified gadget.

**Signature**  gadget-client-setter client gadget => client

**Parameters**

- client – An instance of type <object>.
- gadget – An instance of type <gadget>.

**Values**

- client – An instance of type <object>.

**Discussion**

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>, 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

gadget-command Generic function
Returns the command associated with the specified gadget.

Signature  gadget-command gadget => command

Parameters
• gadget – An instance of type <gadget>.

Values
• command – An instance of type false-or(<frames.htm#40934>).

Discussion
Returns the command associated with gadget.

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
• gadget-command-setter

gadget-command-setter Generic function
Sets the command of the specified gadget.

Signature  gadget-command-setter command gadget => command

Parameters
• command – An instance of type false-or(<frames.htm#40934>).
• gadget – An instance of type <gadget>.

Values
• command – An instance of type false-or(<frames.htm#40934>).

Discussion
Sets the command of the specified gadget.

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
• gadget-command

gadget-default? Generic function
Returns true if the specified gadget is the default gadget in a frame.

Signature  gadget-default? gadget => default?

Parameters
• gadget – An instance of type <gadget>.

Values
• default? – An instance of type <boolean>.
Discussion

Returns true if the specified gadget is the default gadget for the frame it is part of.

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?-setter


gadget-default?-setter Generic function

Toggles whether the specified button is the default for the current frame.

Signature  
gadget-default?-setter default? button => default?

Parameters

- default? – An instance of type <boolean>.
- button – An instance of type <button>.

Values

- default? – An instance of type <boolean>.

Discussion

If default? is true, button becomes the default gadget for the current frame. If default? is #f, button is not the default gadget for the current frame, regardless of any previous value the gadget-default?-setter 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?


gadget-documentation Generic function

Returns the documentation string for the specified gadget.

Signature  
gadget-documentation gadget => documentation

Parameters

- gadget – An instance of type <gadget>.

Values

- documentation – An instance of type false-or(<string>).

Discussion

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 tool-tip (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`

`gadget-documentation-setter` *Generic function*

Sets the documentation string for the specified gadget.

**Signature**

```
gadget-documentation-setter documentation gadget => documentation
```

**Parameters**

- `documentation` – An instance of type `<string>`.
- `gadget` – An instance of type `<gadget>`.

**Values**

- `documentation` – An instance of type `<string>`.

**Discussion**

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 tool-tip (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`
- `<status-bar>`

`gadget-enabled?` *Generic function*

Returns true if the gadget is enabled.

**Signature**

```
gadget-enabled? gadget => enabled?
```

**Parameters**

- `gadget` – An instance of type `<gadget>`.

**Values**

- `enabled?` – An instance of type `<boolean>`.

**Discussion**

Returns true if `gadget` 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

```
gadget* := contain(make
  (<radio-box>,
   items: range(from: 0, to: 20)));
gadget-enabled?(*gadget*);
```

See also

- `<gadget>`
- `gadget-enabled?-setter`

`gadget-enabled?-setter` Generic function
Toggles the enabled state of the specified gadget.

**Signature**
gadget-enabled?-setter enabled? gadget => enabled?

**Parameters**
- `enabled?` – An instance of type `<boolean>`.
- `gadget` – An instance of type `<gadget>`.

**Values**
- `enabled?` – An instance of type `<boolean>`.

**Discussion**
Causes gadget to become active (that is, available for input) or inactive, by toggling its enabled state. If `enabled?` is true, then `gadget` is enabled, otherwise, `gadget` 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

```
gadget* := contain(make
  (<radio-box>,
   items: range(from: 0, to: 20)));
gadget-enabled?(*gadget*) := #f;
```

See also

- `<gadget>`
- `gadget-enabled?`

`gadget-id` Generic function
Returns the ID of the specified gadget.

**Signature**
gadget-id gadget => id

**Parameters**
- `gadget` – An instance of type `<gadget>`.

**Values**
- `id` – An instance of type `<object>`.
Discussion

Returns 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. 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

```dylan
*gadget* := contain(make(<button>, id: #test,
       label: "Test"));
gadget-id(*gadget*);
```

See also

- gadget-id-setter
- gadget-value
- <tab-control>

gadget-id-setter Generic function

Sets the ID of the specified gadget.

Signature  gadget-id-setter id gadget => id

Parameters

- id – An instance of type <object>.
- gadget – An instance of type <gadget>.

Values

- id – An instance of type <object>.

Discussion

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

```dylan
*gadget* := contain(make(<button>, id: #test,
       label: "Test"));
gadget-id(*gadget*) := #test-two;
gadget-id(*gadget*);
```

See also

9.11. DUIM-Gadgets Module
gadget-items Generic function
Returns the items for the specified gadget.

Signature  gadget-items gadget => items

Parameters
• gadget – An instance of type <collection-gadget>.

Values
• items – An instance of type <sequence>. Default value: #[].

Discussion Returns the items for gadget. 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.

```duim
*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:

```duim
gadget-items(*g*);
```

This returns the symbol: #(#"one", #"two", #"three").

See also
• gadget-items-setter
• gadget-label-key
• gadget-selection
• gadget-value-key

gadget-items-setter Generic function
Sets the items for the specified gadget.

Signature  gadget-items-setter items gadget => items

Parameters
• items – An instance of type <sequence>.
• gadget – An instance of type <collection-gadget>.

Values
• items – An instance of type <sequence>.

Discussion Sets the items for gadget to the items specified by items.
Example

```scheme
*gadget* := contain(make
    (<radio-box>,
     items: range(from: 0, to: 20)));

gadget-items(*gadget*) := range(from: 0, to: 15);
```

See also

- `gadget-items`

`gadget-key-press-callback` **Generic function**

Returns the key-press callback for the specified gadget.

**Signature**

```scheme
gadget-key-press-callback gadget => key-press-callback
```

**Parameters**

- `gadget` – An instance of type `<collection-gadget>`.

**Values**

- `key-press-callback` – An instance of type `false-or(<frames.htm#40934>, <function>)`.

**Discussion**

Returns the key-press callback for `gadget`. 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

- `gadget-key-press-callback-setter`
- `<list-control>`
- `<tab-control>`
- `<table-control>`
- `<tree-control>`

`gadget-key-press-callback-setter` **Generic function**

Sets the key-press callback for the specified gadget.

**Signature**

```scheme
gadget-key-press-callback-setter key-press-callback gadget * => *key-press-callback
```

**Parameters**

- `key-press-callback` – An instance of type `false-or(<frames.htm#40934>, <function>)`.
- `gadget` – An instance of type `<collection-gadget>`.

**Values**

- `key-press-callback` – An instance of type `false-or(<frames.htm#40934>, <function>)`. 

Discussion

Sets the key-press callback for gadget. 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

- gadget-key-press-callback
- <list-control>
- <tab-control>
- <table-control>
- <tree-control>

**gadget-label** Generic function

Returns the label for the specified gadget.

Signature  
gadget-label gadget => label

Parameters

- gadget – An instance of type <gadget>.

Values

- label – An instance of type type-union(<string>, <image>).

Discussion  
Returns the label for gadget.

Example

```scheme
*gadget* := contain(make(<button>, label: "Hello"));
gadget-label(*gadget*);
```

See also

- gadget-label-key
- gadget-label-setter

**gadget-label-key** Generic function

Returns the function that is used to compute the label for the items in the specified gadget.

Signature  
gadget-label-key gadget => label-key

Parameters

- gadget – An instance of type <collection-gadget>.

Values

- label-key – An instance of type <function>.

Discussion  
Returns the function that is used to compute the labels for the items in gadget. 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.

```plaintext
*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:

```plaintext
gadget-label-key(*gadget*);
```

See also

- `gadget-label`
- `gadget-label-setter`
- `gadget-value-key`

`gadget-label-setter` Generic function

Sets the label for the specified gadget.

Signature  gadget-label-setter label gadget => label

Parameters

- `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>)`.

Discussion  Sets the label for `gadget` to `label`. The `label` must be a string, 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

```plaintext
*gadget* := contain(make(<button>, label: "Hello"));
gadget-label(*gadget*) := "Hello world";
```

See also

- `gadget-label`
- `gadget-label-key`

`gadget-mnemonic` Generic function

Returns the mnemonic for the specified gadget.

Signature  gadget-mnemonic gadget => mnemonic

Parameters

- `gadget` – An instance of type `<gadget>`.
Values

- **mnemonic** – An instance of type `false-or(<character>)`.

Discussion  Returns the mnemonic for `gadget`. 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

- `gadget-accelerator`
- `gadget-mnemonic-setter`

**gadget-mnemonic-setter** Generic function

Sets the mnemonic for the specified gadget.

**Signature**  `gadget-mnemonic-setter mnemonic gadget => mnemonic`

**Parameters**

- **mnemonic** – An instance of type `false-or(<character>)`.
- **gadget** – An instance of type `<gadget>`.

**Values**

- **mnemonic** – An instance of type `false-or(<character>)`.

Discussion  Sets the mnemonic for `gadget` to `mnemonic`. 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

- `gadget-accelerator-setter`
- `gadget-mnemonic`

**gadget-orientation** Generic function

Returns the orientation of the specified gadget.

**Signature**  `gadget-orientation gadget => orientation`

**Parameters**

- **gadget** – An instance of type `<gadget>`.

**Values**

- **orientation** – An instance of type `one-of(#"horizontal", #"vertical", #"none")`.

**Discussion**  Returns the orientation of `gadget`: either horizontal or vertical.

**Example**  The following code creates a vertical row of buttons:

```scheme
*buttons* := contain(make(<button-box>,
   selection-mode: #"multiple",
   orientation: #"vertical",
   items: range(from: 0, to: 5)));
```

The orientation can be returned as follows:

```scheme
gadget-orientation(*buttons*);
```
gadget-popup-menu-callback Generic function
    Returns the popup menu callback of the specified gadget.

    **Signature**  gadget-popup-menu-callback gadget => popup-menu-callback

    **Parameters**
    - gadget – An instance of type `<gadget>`.

    **Values**
    - popup-menu-callback – An instance of type `<function>`.

    **Discussion**  Returns the popup menu callback of gadget. 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**
    - gadget-popup-menu-callback-setter

  gadget-popup-menu-callback-setter Generic function
  Sets the popup menu callback of the specified gadget.

  **Signature**  gadget-popup-menu-callback-setter popup-menu-callback gadget => popup-menu-callback

  **Parameters**
  - popup-menu-callback – An instance of type `<function>`.
  - gadget – An instance of type `<gadget>`.

  **Values**
  - popup-menu-callback – An instance of type `<function>`.

  **Discussion**  Sets the popup menu callback of gadget to function. 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**
  - gadget-popup-menu-callback

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

  **Signature**  gadget-ratios splitter => ratios

  **Parameters**
  - splitter – An instance of type `<splitter>`.

  **Values**
  - ratios – An instance of type false-or(<sequence>).

  gadget-ratios-setter Generic function
  Sets the ratios of the windows in splitter. This generic function lets you set the position of a splitter.

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

  **Parameters**
  - ratios – An instance of type false-or(<sequence>).
  - splitter – An instance of type `<splitter>`.
Values

• ratios – An instance of type false-or(<sequence>).

Discussion  Set ratios to #f if you do not care what ratios are used.

gadget-read-only? Generic function
Returns true if the gadget is editable.

Signature  gadget-read-only? gadget => read-only?

Parameters

• gadget – An instance of type <gadget>.

Values

• read-only? – An instance of type <boolean>.

Discussion  Returns true if gadget is read-only. The read-only attribute of a gadget is of most use with text gadgets.

See also

• gadget-enabled?

gadget-scrolling-horizontally? Generic function
Returns true if the specified gadget has an associated horizontal scroll bar.

Signature  gadget-scrolling-horizontally? gadget => horizontal?

Parameters

• gadget – An instance of type <gadget>.

Values

• horizontal? – An instance of type <boolean>.

Discussion  Returns true if the gadget has an associated horizontal scroll bar, false otherwise.

See also

• gadget-scrolling-vertically?

gadget-scrolling-vertically? Generic function
Returns true if the specified gadget has an associated vertical scroll bar.

Signature  gadget-scrolling-vertically? gadget => vertical?

Parameters

• gadget – An instance of type <gadget>.

Values

• vertical? – An instance of type <boolean>.

Discussion  Returns true if the gadget has an associated vertical scroll bar, false otherwise.

See also

• gadget-scrolling-horizontally?

gadget-selection Generic function
Returns the currently selected items of the specified gadget.

Signature  gadget-selection gadget => selection

Parameters
• **gadget** – An instance of type `<collection-gadget>`.

**Values**

• **selection** – An instance of type `limited(<sequence>, of: <integer>)`. Default value: `#[]`.

**Discussion**

Returns the keys for the currently selected items of `gadget`. Generally, you should use `gadget-value` to return the selected item, rather than `gadget-selection`, 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 value key of the gadget on each selected item in the gadget.

**Example** Create a radio box as follows:

```daml
*radio* := contain(make(<radio-box>,
   items: range(from: 0, to: 5)));
```

Select one of the items in the radio box. This selection can be returned with:

```daml
gadget-selection(*radio*);
```

**See also**

• gadget-items  
• gadget-selection-mode  
• gadget-selection-setter  
• gadget-value

**gadget-selection-mode Generic function**

Returns the type of selection for the specified gadget.

**Signature** `gadget-selection-mode gadget => selection-mode`

**Parameters**

• **gadget** – An instance of type `<collection-gadget>`.

• **selection-mode** – An instance of type `one-of(#"single", #"multiple", #"none")`.

**Discussion** Returns the selection mode for `gadget`. 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 `selection-mode:init-keyword`.

**Example** Create a radio box as follows:

```daml
*radio* := contain(make(<radio-box>,
   items: range(from: 0, to: 5)));
```

The selection mode of the radio box is returned with:
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>
- gadget-selection
- gadget-selection-setter
- <list-box>

**gadget-selection-setter** Generic function

Sets the selection of the specified gadget.

**Signature**
gadget-selection-setter selection gadget #key do-callback? => selection

**Parameters**

- **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>).

**Discussion**

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 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
- gadget-selection-mode
- gadget-value-setter
gadget-slug-size Generic function
Returns the slug size of the specified gadget.

Signature  gadget-slug-size gadget => slug-size

Parameters
  • gadget – An instance of type <scroll-bar>.

Values
  • slug-size – An instance of type <real>.

Discussion
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
  • gadget-value-range

gadget-slug-size-setter Generic function
Sets the slug size of the specified gadget.

Signature  gadget-slug-size-setter slug-size gadget => slug-size

Parameters
  • slug-size – An instance of type <real>.
  • gadget – An instance of type <gadget>.

Values
  • slug-size – An instance of type <real>.

Discussion
Sets the slug size of gadget. The value should use 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

gadget-test Generic function
Returns the test function for the specified gadget.

Signature  gadget-test gadget => gadget-test

Parameters
• gadget – An instance of type \texttt{<collection-gadget>}.

Values
• \texttt{gadget-test} – An instance of type \texttt{<function>}.

Discussion  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
Returns the text for the specified gadget.

Signature  \texttt{gadget-text gadget \Rightarrow gadget-text}

Parameters
• \texttt{gadget} – An instance of type \texttt{<text-gadget>}.

Values
• \texttt{gadget-text} – An instance of type \texttt{<string>}.

Discussion  Returns the text for the specified gadget.

Example  First, create and display a text field by typing the following into an interactor:

\begin{verbatim}
*g* := contain(make(<text-field>,
    value-type: <integer>));
\end{verbatim}

Next, type something into the text field. You can return the text string that you just typed with the following form:

\begin{verbatim}
gadget-text(*g*);
\end{verbatim}

See also
• \texttt{gadget-text-setter}
• \texttt{<text-gadget>}

**gadget-text-setter** Generic function
Sets the text for the specified gadget.

Signature  \texttt{gadget-text gadget-text gadget \Rightarrow gadget-text}

Parameters
• \texttt{gadget-text} – An instance of type \texttt{<string>}.
• \texttt{gadget} – An instance of type \texttt{<text-gadget>}.

Values
• \texttt{gadget-text} – An instance of type \texttt{<string>}.

Discussion  Sets the text for the specified gadget.

Example  First, create and display a text field by typing the following into an interactor:

\begin{verbatim}
*g* := contain(make(<text-field>,
    value-type: <integer>));
\end{verbatim}

Next, set the value of the text field with the following form:

\begin{verbatim}
gadget-text-setter("Hello world", *g*);
\end{verbatim}
gadget-value Generic function
Returns the gadget value of the specified gadget.

Signature  gadget-value gadget => gadget-value

Parameters
  • gadget – An instance of type <value-gadget>.

Values
  • gadget-value – An instance of type <object>.

Discussion
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>
  • gadget-id
  • gadget-value-key
  • gadget-value-range
  • gadget-value-setter
• **gadget-value-type**

**gadget-value-changed-callback** Generic function

Returns the value-changed callback of the specified gadget.

**Signature**
gadget-value-changed-callback gadget => value-changed-callback

**Parameters**

• **gadget** – An instance of type `<value-gadget>`.

**Values**

• **value-changed-callback** – An instance of type false-or(<function>).

**Discussion**

Returns the value-changed callback of `gadget`. This is the callback function that is called once the gadget value of `gadget` has been changed.

The value-changed callback function is invoked with one argument, the gadget.

If `gadget-value-changed-callback` returns `#f`, there is no value changed callback for `gadget`.

**See also**

• **gadget-value-changed-callback-setter**

**gadget-value-changed-callback-setter** Generic function

Sets the value-changed callback of the specified gadget.

**Signature**
gadget-value-changed-callback-setter callback gadget => callback

**Parameters**

• **callback** – An instance of type false-or(<function>).

• **gadget** – An instance of type `<gadget>`.

**Values**

• **callback** – An instance of type false-or(<function>).

**Discussion**

Sets the value-changed callback of `gadget` to `function`. This is the callback function that is called once the gadget value of `gadget` has been changed.

The value-changed callback function is invoked with one argument, the gadget.

**See also**

• **gadget-value-changed-callback**

**gadget-value-changing-callback** Generic function

Returns the value changing callback of the specified gadget.

**Signature**
gadget-value-changing-callback gadget => value-changing-callback

**Parameters**

• **gadget** – An instance of type `<gadget>`.

**Values**

• **value-changing-callback** – An instance of type `<function>`.
Discussion Returns the function that will be called when the value of gadget is in the process of changing, such as when a slider is being dragged. The function will be invoked with a two arguments, gadget and the new value.

See also
• gadget-value-changing-callback-setter

**gadget-value-changing-callback-setter** Generic function
Sets the value-changing callback of the specified gadget.

**Signature**
gadget-value-changing-callback-setter value-changing-callback gadget => value-changing-callback

**Parameters**
• value-changing-callback – An instance of type <function>.
• gadget – An instance of type <gadget>.

**Values**
• value-changing-callback – An instance of type <function>.

Discussion Sets the function that will be called when the value of gadget is in the process of changing, such as when a slider is being dragged. The function will be invoked with a two arguments, gadget and the new value.

See also
• gadget-value-changing-callback

**gadget-value-key** Generic function
Returns the function that is used to calculate the gadget value of the specified gadget.

**Signature**
gadget-value-key gadget => value-key

**Parameters**
• gadget – An instance of type <collection-gadget>.

**Values**
• value-key – An instance of type <function>. Default value: identity.

Discussion Returns the function that is used to calculate the gadget value of gadget, 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
- gadget-value

**gadget-value-range** Generic function

Returns the range of values for the specified gadget.

**Signature**

```lisp
gadget-value-range gadget => range
```

**Parameters**

- `gadget` – An instance of type `<value-range-gadget>`.

**Values range**

An instance of type `<range>`.

**Discussion**

Returns the range of values for `gadget`. The value range is the elements represented by the range specified for `gadget`.

**Note:** The value range is not simply the difference between the maximum and minimum values in the range. Consider the following range:

```lisp
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:

```lisp
*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:

```lisp
gadget-value-range(*slider*);
```

which in this case returns `{range -20 through 20, by 5}`.

See also

- gadget-slug-size
- gadget-value
- gadget-value-range-setter

**gadget-value-range-setter** Generic function

Sets the range of values for the specified gadget.

**Signature**

```lisp
gadget-value-range-setter range gadget => range
```

**Parameters**

- `range` – An instance of type `<range>`.
• **gadget** – An instance of type `<value-range-gadget>`.

**Values**

• **range** – An instance of type `<range>`.

**Discussion** 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:

```lisp
*slider* := contain(make(<slider>));
```

You can specify the range of this gadget by executing the following:

```lisp
gadget-value-range(*slider*) :=
  (range {from: -20 to: 20, by: 5});
```

**See also**

• **gadget-value-range**

**gadget-value-setter** Generic function

Sets the gadget value of the specified gadget.

**Signature** gadget-value-setter value gadget #key do-callback? => value

**Parameters**

• **value** – An instance of type `<object>`.
• **gadget** – An instance of type `<value-gadget>`.
• **do-callback?** – An instance of type `<boolean>`. Default value: #f.

**Values**

• **value** – An instance of type `<object>`.

**Discussion**

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:

```lisp
*radio* := contain(make(<radio-button>,
  label: "Radio"));
```

The gadget value of *radio* can be set with either of the following:

```lisp
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**
gadget-value-type  Generic function
Returns the type of the gadget value for the specified gadget.

**Signature**  
gadget-value-type gadget => type

**Parameters**
- **gadget** – An instance of type `<value-gadget>`.

**Values**
- **type** – An instance of type `<type>`.

**Discussion**  
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.

```duim
*numeric* := contain(make(<text-field>,
    value-type: <integer>));
```

Evaluating the following code confirms the gadget value type to be the class `<integer>`.

```duim
gadget-value-type(*numeric*);
```

**See also**
- **gadget-value**

---

gadget-x-alignment  Generic function
Returns the horizontal alignment of the specified gadget.

**Signature**  
gadget-x-alignment gadget => alignment

**Parameters**
- **gadget** – An instance of type `<gadget>`.

**Values**
- **alignment** – An instance of type `one-of(#"left", #"right", #"center")`.

**Discussion**  
Returns the horizontal alignment of `gadget`. You can only set the horizontal alignment of a gadget when first initializing that gadget, using the `x-alignment: init-keyword`.

**See also**
- **gadget-y-alignment**

---

gadget-y-alignment  Generic function
Returns the vertical alignment of the specified gadget.

**Signature**  
gadget-y-alignment gadget => alignment

**Parameters**
- **gadget** – An instance of type `<gadget>`.

**Values**
- **alignment** – An instance of type `one-of(#"top", #"bottom", #"center")`.

**Discussion**  
Returns the vertical alignment of `gadget`. You can only set the vertical alignment of a gadget when first initializing that gadget, using the `y-alignment: init-keyword`.

**See also**
<group-box> Open Abstract Instantiable Class
The class of gadgets that group their children using a labelled border.

**Superclasses** `<gadget>`

**Init-Keywords**

- `label` – An instance of type `<label>`.
- `label-position` – An instance of type `one-of(#"top", #"bottom")`. Default value: `#"top"`.

**Discussion**

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.

![Fig. 9.11: A group box](image)

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.

**Example**

```dylan
contain(make(<group-box>,
    child: make(<radio-box>,
        items: #(1,2,3,4),
        orientation: #"vertical"),
    label: "Select integer:"));
```

See also

- `<border>`
- `<check-box>`
- `<push-box>`
- `<radio-box>`

**item-object** Generic function

Returns the Dylan object representing an item in a list or table control.

**Signature** `item-object item => object`

**Parameters**

- `item` – An instance of type `type-union(<list-item>, <table-item>)`.

**Values**

- `object` – An instance of type `<object>`.
**Discussion**  Returns the Dylan object representing an item in a list or table control.

<label> Open Abstract Instantiable Class
The class of label gadgets.

**Superclasses** <gadget>

**Init-Keywords**

- **label** – An instance of type type-union(<string>, <image>).

**Discussion**

The class of label gadgets.

The **label**: 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.

Internally, this class maps into the Windows static control.

**Operations**

- **gadget-label**
- **gadget-label-setter**
- <frames.htm#74637>
- <frames.htm#10131>
- <frames.htm#68823>
- <frames.htm#14565>

**Example**

```dylan
contain(make(<label>, label: "Hello"));
```

**See also**

labelling

**labelling Statement Macro**

Creates the specified sheet and assigns a label to it.

**Macro Call**

```dylan
labelling ([*options* ]) {*pane* } end
```

**Parameters**

- **options** – Dylan arguments bnf.
- **pane** – A Dylan expression bnf.

**Discussion**

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>

<list-box> Open Abstract Instantiable Class

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".

**Discussion**

![List Box](image)

Fig. 9.12: 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.

**Example** The following creates a list of three items, without scroll bars.

```duim
*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>

• <list-item>
**<list-control> Open Abstract Instantiable Class**  
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".
- **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(<frames.htm#40934>, <function>)`.

**Discussion**

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`, for a fuller description of key-press callbacks.

Internally, this class maps into the Windows list view control.

**Operations**
- `add-item`
- `find-item`
- `list-control-view`
- `list-control-view-setter`
- `make-item`
- `remove-item`

**See also**
- `add-item`
- `list-control-view`
- `make-item`
- `remove-item`

`list-control-icon-function` **Generic function**
Returns the icon function for the specified list control.

**Signature** `list-control-icon-function list-control => icon-function`

**Parameters**
- `list-control` – An instance of `<list-control>`.

**Values**
- `icon-function` – An instance of type `<function>`.

**Discussion**
Returns the icon-function for `list-control`. 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 `<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.

Note that, unlike tree controls, the icon function for a list control can be changed once the list control has been created.

**See also**
- `<list-control>`
- `list-control-icon-function-setter`

`list-control-icon-function-setter` **Generic function**
Sets the icon function for the specified list control.

**Signature** `list-control-icon-function-setter icon-function list-control => icon-function`

**Parameters**
- `icon-function` – An instance of type `<function>`.
- `list-control` – An instance of `<list-control>`.
Values

- **icon-function** – An instance of type `<function>`.

Discussion

Sets the icon-function for *list-control*. 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 `<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.

Note that, unlike tree controls, the icon function for a list control can be changed once the list control has been created.

See also

- `<list-control>`
- `list-control-icon-function`

*<list-control-view>* Type

The type of possible views for a list control

**Equivalent** one-of(#"small-icon", #"large-icon", #"list")

Discussion

This type represents the acceptable values for the view arguments to operators of `<list-control>`. You should not attempt to redefine this type in any way.

There are three possible values, corresponding to the view options that will be familiar to most users of GUI-based operating systems:

- #"small-icon" Displays each item in the list control using a small icon to the left of the item. Items are arranged horizontally.
- #"large-icon" Displays each item in the list control using a large icon to the left of the item. Items are arranged horizontally.
- #"list" Displays each item in the list control using a small icon to the left of the item. Items are arranged vertically in one column.

See also

- :class:`<list-control>`
- `<list-control-view>`
- `<table-control-view>`

*list-control-view* Generic function

Returns the view for the specified list control.

**Signature**  list-control-view list-control => view

**Parameters**

- **list-control** – An instance of `<list-control>`.

**Values**

- **view** – An instance of type `<list-control-view>`.
**Discussion** 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>.

**Example** Given a list control created with the following code:

```plaintext
*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:

```plaintext
list-control-view(*list*);
```

**See also**

- <list-control>
- <list-control-view>
- list-control-view-setter

**list-control-view-setter** Generic function

Sets the view for the specified list control.

**Signature** list-control-view-setter view list-control => view

**Parameters**

- **view** – An instance of type <list-control-view>.
- **list-control** – An instance of <list-control>.

**Values**

- **view** – An instance of type <list-control-view>.

**Discussion** 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>.

**Example** Given a list control created with the following code:

```plaintext
*list* := contain(make(<list-control>,
    items: #("One",
    "Two",
    "Three"));
```

The list control view may be specified with:

```plaintext
list-control-view(*list*) := #"view";
```

**See also**

- <list-control>
- <list-control-view>
• list-control-view

<list-item> Open Abstract Instantiable Class

The class that represents an item in a list control.

Superclasses <object>

Init-Keywords

• object – An instance of type <object>. Default value: #f.

Discussion The class that represents an item in a list control.

Operations

• add-item
• item-object
• remove-item

See also

• <list-control>
• <table-item>

make-item Generic function

Creates an item which can be inserted in the specified list control or table control.

Signature make-item list-or-table object #key frame-manager => item

Parameters

• list-or-table – An instance of type-union(<list-control>, <table-control>).

• object – An instance of type <object>.

• frame-manager (#key) – An instance of type <frame-manager>.

Values

• item – An instance of type <list-item>.

Discussion

Creates an item that represents object which can be inserted in the specified list-or-table. To insert the item in the list control or table control, add-item is used. You would not normally call make-item explicitly: just use add-item and the item is created automatically before it is added to the list or table control.

If the frame-manager argument is specified, then this is used instead of the default frame manager.

See also

• add-item
• find-item
• <list-control>
• <list-item>
• remove-item
make-menu-from-items Generic function
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

Parameters
  • framem – An instance of type <frame-manager>.
  • items – An instance of type <sequence>.
  • owner(#key) – An instance of type <sheet>.
  • title(#key) – An instance of type <string>'.
  • label-key(#key) – An instance of <function>. Default value: identity.
  • value-key(#key) – An instance of <function>. Default value: identity.
  • foreground(#key) – An instance of type false-or(<ink>). Default value: #f.
  • background(#key) – An instance of type false-or(<ink>). Default value: #f.
  • text-style(#key) – An instance of type <text-style>.

Values
  • menu – An instance of type <menu>.

Discussion
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’, or ‘gadget-value-key’.

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

make-node Generic function
Creates a node which can be inserted in the specified tree control.

Signature  make-node tree object #key #all-keys => node

Parameters
  • tree – An instance of <tree-control>.
  • object – An instance of type <object>.
Values

- **node** – An instance of type `<tree-node>`.

Discussion

Creates a node that represents object which can be inserted in the specified tree. To insert the item in the tree control, `add-node` is used. You would not normally call `make-node` explicitly: just use `add-node` and the node is created automatically before it is added to the tree control.

See also

- `add-node`
- `find-node`
- `remove-node`
- `<tree-control>`

<menu> Open Abstract Instantiable Class

The class of menu gadgets.

Superclasses `<gadget> <multiple-child-composite-pane>`

Init-Keywords

- **update-callback** – An instance of type `false-or(<function>)`.
- **owner** – An instance of type `<sheet>`.
- **mnemonic** – An instance of type `false-or(<character>)`. Default value: `#f`.
- **command** – An instance of `false-or(<frames.htm#40934>)`. Default value: `#f`.

Discussion

The class of menu gadgets.

Support for dynamically modifying the contents of a menu 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 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 `:class:<menu-box>`.

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.

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.

The `command:` 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 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

- `<frames.htm#89020>`
- `choose-from-dialog`
- `choose-from-menu`
Example

The following code creates a menu, *Hello*, that contains a single button, *World*. Notice how using `contain` creates a menu bar for you automatically. You should note that using `display-menu` would not have this effect.

```duim
*menu* := contain(make(<menu>,
    label: "Hello",
    children: vector
        (make(<menu-button>,
            label: "World"))));
```

See also

- `display-menu`
- `make-menu-from-items`

### `<menu-bar>` Open Abstract Instantiable Class

The class of menu bar gadgets.

**Superclasses** `<value-gadget>` `<multiple-child-composite-pane>`

**Init-Keywords**

- `update-callback` – An instance of type `<function>`.

**Discussion**

The class of menu bar gadgets.

Internally, this class maps into the Windows menu control.

**Operations**

- `<frames.htm#63229>`
- `<frames.htm#56600>`

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`:

```duim
*menu* := make(<menu-bar>,
    children: vector(make(<menu>,
        label: "Hello",
        children: vector
            (make(<menu-button>,
                label: "World"))));
```

See also

- `<menu>`

### `<menu-box>` Open Abstract Instantiable Class

A class that groups menu buttons.

**Superclasses** `<collection-gadget>`

**Init-Keywords**
• **update-callback** – An instance of type `<false-or(<function>)>.

**Discussion**

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.

**Example**

```lisp
*menu-box* := contain(make(<menu-box>,
    items: range
    (from: 0, to: 5)));
```

**See also**

• `<check-menu-box>`
• `<push-menu-box>`
• `<radio-menu-box>`

**<menu-button>** Open Abstract Instantiable Class

The class of all buttons that can appear in menus.

**Superclasses** `<button>`

**Init-Keywords**

• **update-callback** – An instance of type `<function>`.

**Discussion**

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>`

Internally, this class maps into the menu item Windows control.

**Example**

```lisp
contain
(make(<menu-button>, label: "Hello",
    activate-callback:
    method (gadget)
    notify-user
    (format-to-string
    ("Pressed button %s", gadget),
    owner: gadget) end));
```
See also

- `<check-menu-button>`
- `gadget-accelerator`
- `<menu-box>`
- `<push-menu-button>`
- `<radio-menu-button>`

**menu-owner** Generic function

Returns the sheet that owns the specified menu.

**Signature**

`menu-owner menu => sheet`

**Parameters**

- `menu` – An instance of type `<menu>`.

**Values**

- `sheet` – An instance of type `<sheet>`.

**Discussion**

Returns the sheet that owns `menu`, that is, the sheet in which `menu` 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

Returns the children of the specified node in a tree control.

**Signature**

`node-children tree-node => children`

**Parameters**

- `tree-node` – An instance of type `<tree-node>`.

**Values**

- `children` – An instance of type `limited(<sequence>, of: <tree-node>).`

**Discussion**

Returns the children of `tree-node` in a tree control.

See also

- `node-children-setter`
- `node-parents`
- `tree-control-children-generator`
- `<tree-node>`

**node-children-setter** Generic function

Sets the children of the specified node in a tree control.

**Signature**

`node-children-setter children tree-node => children`

**Parameters**

- `children` – An instance of type `limited(<sequence>, of: <tree-node>).`.
- `tree-node` – An instance of type `<tree-node>`.

**Values**
• children – An instance of type limited(<sequence>, of: <tree-node>).

Discussion Sets the children of tree-node in a tree control.

See also
• node-children
• node-parents
• tree-control-children-generator
• <tree-node>

node-expanded? Generic function
Returns true if the specified node is expanded in a tree control.

Signature node-expanded? tree-node => expanded?

Parameters
• tree-node – An instance of type <tree-node>.

Values
• expanded? – An instance of type <boolean>.

Discussion Returns true if tree-node is expanded in a tree control, so that its children are displayed in the tree control.

See also
• <tree-node>

node-object Generic function
Returns the object that the specified node in a tree control represents.

Signature node-object tree-node => object

Parameters
• tree-node – An instance of type <tree-node>.

Values
• object – An instance of type <object>.

Discussion Returns the object that tree-node represents.

See also
• <tree-node>

node-parents Generic function
Returns the parents of the specified node in a tree control.

Signature node-parents tree-node => parents

Parameters
• tree-node – An instance of type <tree-node>.

Values
• parents – An instance of type <sequence>.

Discussion Returns the parents of tree-node in a tree control.

See also
node-state Generic function
Returns the state of the specified node in a tree control.

Signature  node-parents tree-node => state

Parameters
  • tree-node – An instance of type <tree-node>.

Values
  • parents – An instance of type one-of(#"expanded", #"contracted", #f).

Discussion  Returns the state of tree-node 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  
  • node-expanded?
  • <tree-node>

<option-box> Open Abstract Instantiable Class
The class of option boxes.

Superclasses <collection-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".

Discussion

Fig. 9.13: 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.

See also  
  • <combo-box>

<page> Open Abstract Instantiable Class
The class that represents a page in a tab control.
Superclasses  <gadget>
Init-Key-words
  • label – An instance of type type-union(<string>, <image>).
Discussion
  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
  • <frames.htm#88015>
  • <frames.htm#89408>
See also
  • <frames.htm#93333>
  • <tab-control-page>
  • <frames.htm#87607>

<password-field> Open Abstract Instantiable Class
  The class of text fields that do not echo typed text.
Superclasses  <text-field>
Discussion
  The class of text fields that do not echo typed text. This class of gadgets are very similar in appearance to the <text-field> 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.
  Internally, this class maps into the Windows single-line edit control with ES-PASSWORD style.
Example

  *pass* := contain(make(<password-field>));

See also
  <text-field>

<progress-bar> Open Abstract Instantiable Class
  The class of progress bar windows.
Superclasses  <value-range-gadget>
Init-Key-words
  • orientation – An instance of type one-of(#"horizontal", "vertical"). Default value: #"horizontal".
Discussion
  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.

**Example** The following code creates an “empty” progress bar:

```plaintext
*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:

```plaintext
for (i from 0 to 100) gadget-value(*prog*) := i end;
```

See also

• `<slider>`

**<push-box> Class**

Open

Abstract

Instantiable

The class of grouped push buttons.

**Superclasses** `<button-box>` `<action-gadget>`

**Discussion**

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.

**Example**

```plaintext
*push-box* := contain
    (make(<push-box>,
        items: range(from: 0, to: 5)));
```

See also
<push-button> Open Abstract Instantiable Class
The class of push buttons.

Superclasses <button> <action-gadget>

Init-Keywords

• default? – An instance of type <boolean>. Default value: #f.

Discussion

![Push button](image)

Fig. 9.16: 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.

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>
• <radio-button>

<push-menu-box> Open Abstract Instantiable Class
The class of grouped push buttons in menus.

Superclasses <menu-box> <action-gadget>
Fig. 9.17: The class of grouped push buttons in menus.

Discussion

Internally, this class maps into the menu Windows control.

Example

```
contain(make(<push-menu-box>,
    items: range(from: 0, to: 5)));
```

See also

- `<check-menu-box>`
- `<menu-box>`
- `<radio-menu-box>`

*<push-menu-button>* Open Abstract Instantiable Class

The class of push buttons that appear on menus.

Superclasses `<push-menu-button>`

Init-Keywords


Discussion

Fig. 9.18: 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.

See also

- `<check-menu-button>`
- `<menu-button>`
- `<radio-button>`

*<radio-box>* Open Abstract Instantiable Class

The class of radio boxes, or groups of mutually exclusive radio buttons.

Superclasses `<button-box> <action-gadget>`
Discussion

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.

Example

```
contain(make(<radio-box>, items: #("Yes", "No"),
orientation: "vertical"));
```

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.

```
*radio-box* := contain
(make(<radio-box>,
items: #(1, 2, 3, 4, 5),
orientation: "vertical",
label-key:
method (item)
format-to-string("===%d===,
item) end));
```

See also

- `<check-box>`
- `<group-box>`
- `<push-box>`

<radio-button> Open Abstract Instantiable Class

The class of radio buttons.

Superclasses `<button>` `<action-gadget>`

Discussion

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 :class:`<radio-box>` gadget.

Internally, this class maps into the radio button Windows control.

Example

```
contain(make(<radio-button>, label: "Hello"));
```

See also

- `<button>`
- `<check-button>`
- `<menu-button>`
- `<radio-box>`

<radio-menu-box> Open Abstract Instantiable Class

The class of grouped radio buttons that can appear in menus.
Superclasses `<menu-box> <action-gadget>`

Discussion

The class of grouped radio buttons that can appear in menus.

![Radio Menu Box](image)

Fig. 9.19: A radio menu box

Internally, this class maps into the menu Windows control.

**Example** The following example creates a menu that shows an example of a radio menu box, as well as several other menu gadgets.

```plaintext
contain(make(<menu>,
    label: "Hello...",
    children: vector
    (make(<menu-button>,
        label: "World"),
    make(<menu-button>,
        label: "Bonzo"),
    make(<radio-menu-box>,
        items: #("You", "All", "Everyone")),
    make(<menu>,
        label: "Others",
        children: vector
    (make(<check-menu-box>,
        items: #(1, 2, 3))
)))));
```

See also

- `<menu-box>`
- `<push-menu-box>`
- `<radio-menu-button>`

**<radio-menu-button> Open Abstract Instantiable Class**

The class of radio buttons that can appear in menus.

Superclasses `<menu-button>`

Discussion

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 `:class:<radio-menu-box>` gadget.

Internally, this class maps into the menu radio item Windows control.

**Example**
See also

- `<menu-button>`
- `<push-menu-button>`
- `<radio-menu-button>`

**remove-column** Generic function

Removes a column from the specified table.

**Signature** remove-column table index =>

**Parameters**

- **table** – An instance of type `<table-control>`.
- **index** – An instance of type `<integer>`.

**Discussion** Removes a column from `table`.

See also

- `add-column`

**remove-item** Generic function

Removes an item from a list control or table control.

**Signature** remove-item list-or-table item => ()

**Parameters**

- **list-or-table** – An instance of `type-union(<list-control>, <table-control>)`.
- **item** – An instance of type `:class'<list-item>'`.

**Discussion** Removes `item` from `list-or-table`.

See also

- `add-item`
- `find-item`
- `<list-control>`
- `<list-item>`
- `make-item`
- `<table-control>`
- `<table-item>`

**remove-node** Generic function

Removes a node from a tree control.

**Signature** remove-node tree node => ()

**Parameters**

- **tree** – An instance of `<tree-control>`.
- **node** – An instance of type `<tree-node>`.

**Discussion** Removes `node` from `tree`.
See also

- add-node
- find-node
- make-node
- <tree-control>

<scroll-bar> Open Abstract Instantiable Class

The class of scroll bars.

**Superclasses** <value-range-gadget>

**Init-Keywords**

- **orientation** – An instance of type one-of(#"horizontal", #"vertical", #"none"). Default value: #"none".
- **value-changing-callback** – An instance of type <function>.
- **value-changed-callback** – An instance of type <function>.
- **slug-size** – An instance of type <real>.

**Discussion**

The instantiable class that implements an abstract scroll bar.

The **orientation**: init-keyword defines whether the scroll bar is horizontal or vertical.

The **value-changing-callback**: 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.

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>

scrolling Statement Macro
Places scroll bars around the specified DUIM panes, if required.

Macro Call
scrolling ([*options* ]) {*pane* } end

Parameters
• options – Dylan arguments bnf.
• pane – A Dylan expression bnf.

Discussion
Places scroll bars around the DUIM panes created by pane, 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.

Creates pane with scroll bars attached to it, taking into account any of the specified options.

The pane is an expression whose return value is the sheet to which the scroll bars should be attached.

The options can be used to specify the properties of the scroll bars. As well as all the properties of <gadget>, these include a scroll-bars: 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.

The pane is a body of code whose return value is the sheet to which the label should be assigned.

Example

scrolling (scroll-bars: #"vertical")
make(<radio-box>),
  orientation: #"vertical",
  items: range(from: 1, to: 50))
end

See also
• <scroll-bar>
• scroll-position
• set-scroll-position

scroll-position Generic function
Returns the position of the scroll bar slug in the specified sheet.

Signature scroll-position sheet => x y

Parameters
• sheet – An instance of type <sheet>.

Values
• \( x \) – An instance of type \(<\text{integer}\>\).
• \( y \) – An instance of type \(<\text{integer}\>\).

Discussion

Returns the position of the scroll bar slug in \textit{sheet}. Note that this generic function only returns the position of scroll bar slugs that have been created using the \textit{scrolling} macro. It does not work on gadgets with scroll bars defined explicitly.

\textbf{Note:} The Microsoft Windows Interface Guidelines refer to the slug as a \textit{scroll-box}, and the area in which the slug can slide as the \textit{scroll-shaft}. You should be aware of this difference if you are using those guidelines as a reference.

See also

• \textit{scrolling}
• \textit{set-scroll-position}

\textit{<separator>} Open Abstract Instantiable Class

The class of gadgets used as a visual separator.

\textbf{Superclasses} \(<\text{gadget}\>\)

\textbf{Init-Keywords}

• \textit{orientation} – An instance of type \textit{one-of(#"horizontal", #"vertical")}. Default value: #"horizontal".

Discussion

The class of gadgets used as a visual separator.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{separator.png}
\caption{A separator}
\end{figure}

The \textit{orientation}: init-keyword specifies whether the separator is vertical or horizontal.

\textbf{Example} The following example creates a column layout and places two buttons in it, separated with a separator.

\begin{verbatim}
contain(vertically ()
  make(<button>, label: "Hello");
  make(<separator>);
  make(<button>, label: "World")
end);
\end{verbatim}

See also

• \textit{<spacing>}

\textit{set-scroll-position} Generic function

Scrolls the window on the specified sheet.

\textbf{Signature} \textit{set-scroll-position} \textit{sheet} \textit{x y} \textit{=>} ()

\textbf{Parameters}
• **sheet** – An instance of type `<sheet>`.
• **x** – An instance of type `<integer>`.
• **y** – An instance of type `<integer>`.

**Discussion**

Scrolls the window on `sheet` by setting the position of the scroll bar slug. Note that this generic function only sets the position of scroll bar plugs 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 plug as a `scroll-box`, and the area in which the plug can slide as the `scroll-shaft`. You should be aware of this difference if you are using those guidelines as a reference.

**See also**

• `scroll-position`
• `scrolling`

**sheet-viewport** Generic function

Returns the viewport that is clipping the specified sheet.

**Signature**  
`sheet-viewport sheet => viewport`

**Parameters**

• **sheet** – An instance of type `<sheet>`.

**Values**

• **viewport** – An instance of type `false-or('<viewport>')`.

**Discussion**  
Returns the viewport that is clipping `sheet`.

**See also**

• `sheet-viewport-region`
• `<viewport>`

**sheet-viewport-region** Generic function

Returns the sheet region of the specified sheet’s viewport, if it has one.

**Signature**  
`sheet-viewport-region sheet => region`

**Parameters**

• **sheet** – An instance of type `<sheet>`.

**Values**

• **region** – An instance of type `<region>`.

**Discussion**  
Returns the sheet region of `sheet`’s viewport, if it has one. If `sheet` has no viewport, it returns `sheet`’s own region.

**See also**

• `sheet-viewport`
• `<viewport>`
<slider> Open Abstract Instantiable Class
The class of slider gadgets.

Superclasses <value-range-gadget>

Init-Keywords

- min-label – An instance of type type-union(<string>, <image>).
- max-label – An instance of type type-union(<string>, <image>).
- tick-marks – An instance of type false-or(<integer>). Default value: #f
- orientation – An instance of type one-of(#"horizontal", #"vertical"). Default value: #"horizontal".
- value-changing-callback – An instance of type <function>.

Discussion

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.

Example

```
contain(make(<slider>),
    value-range: range(from: -20, to: 20, by: 5));
```

See also

- <scroll-bar>
- <spin-box>

<spacing> Open Abstract Instantiable Class
The class of gadgets that can be used to provide spacing around a sheet.

Superclasses <gadget>
Init-Keywords

- **child** – An instance of type `limited(<sequence> of: <sheet>).
- **thickness** – An instance of type `<integer>`. Default value: 1.

Discussion

The class of gadgets that can be used to provide spacing around a sheet.

The `child`: init-keyword is the sheet or sheets that you are adding spacing around.

The `thickness`: init-keyword specifies the thickness of the spacing required.

It is usually clearer to use the `with-spacing` macro, rather than to create an instance of `<spacing>` explicitly.

**Example**  The following creates a vertical layout containing two buttons separated by a text field that has spacing added to it.

```duim
contain(vertically ()
    make(<button>, label: "Hello");
    make(<spacing>,
        child: make(<text-field>),
        thickness: 10);
    make(<button>, label: "World")
end);```

See also

- `<null-pane>`
- `<separator>`
- `with-spacing`

`<spin-box>` Open Abstract Instantiable Class

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`

Discussion

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.

Example

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

See also

• `<slider>`

<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

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

**Signature**

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

**Parameters**

• `splitter` – An instance of type `<splitter>`.

**Values**

• `function` – An instance of type `<function>`.

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

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

**Signature**

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

**Parameters**

• `function` – An instance of type `<function>`.

• `splitter` – An instance of type `<splitter>`.

**Values**

• `function` – An instance of type `<function>`.

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

Returns the callback invoked when the split box of `splitter` is clicked.
Signature splitter-split-box-callback \texttt{splitter} => \texttt{function}

Parameters

- \texttt{splitter} – An instance of type \texttt{<splitter>}.  

Values

- \texttt{function} – An instance of type \texttt{<function>}.  

\texttt{splitter-split-box-callback-setter} Generic function

Sets the callback invoked when the split box of \texttt{splitter} is clicked.

Signature splitter-split-box-callback-setter \texttt{function splitter} => \texttt{function}

Parameters

- \texttt{function} – An instance of type \texttt{<function>}.  
- \texttt{splitter} – An instance of type \texttt{<splitter>}.  

Values

- \texttt{function} – An instance of type \texttt{<function>}.  

\texttt{<status-bar>} Open Abstract Instantiable Class

The class of status bars.

Superclasses \texttt{<value-range-gadget>}

Init-Keywords

- \texttt{label} – An instance of type \texttt{type-union(<string>, <image>)}.  
- \texttt{label-pane} – An instance of \texttt{false-or(<gadget>)}. Default value: \#f.  
- \texttt{progress-bar?} – An instance of type \texttt{<boolean>}. Default value: \#f.  
- \texttt{progress-bar} – An instance of \texttt{false-or(<progress-bar>)}. Default value: \#f.  
- \texttt{value} – An instance of type \texttt{<object>}.  
- \texttt{value-range} – An instance of type \texttt{<range>}.  

Discussion

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.  

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 \texttt{label:} init-keyword specifies a string or icon that is to be used as a label for the gadget. Alternatively, the \texttt{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?:` to 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**

- `<frames.htm#32720>`
- `<frames.htm#56600>`
- `:gf:status-bar-label-pane`
- `:gf:status-bar-progress-bar`

**Example**  The following creates a basic status bar with the given label, and a progress bar with the given range of values.
See also

- `<frames.htm#12376>`
- `<frames.htm#36830>`
- `gadget-documentation`
- `status-bar-label-pane`
- `status-bar-progress-bar`

**status-bar-label-pane**

Generic function

Returns the gadget that displays the label of the specified status bar.

**Signature**

\[
\text{status-bar-label-pane玩家朋友 status-bar => label}
\]

**Parameters**

- `status-bar` – An instance of type `<status-bar>`.

**Values**

- `label` – An instance of type `false-or(label ')`.

**Discussion**

Returns the gadget that displays the label of `status-bar`.

**Example**

Create a status bar with a label as follows:

\[
*\text{status}: = \text{contain}(\text{make}<\text{status-bar}>,
\text{value-range}: \text{range(from: 0, to: 100),
label: "Status"});
\]

The pane that the label of the status bar is displayed in can be returned with the following call:

\[
\text{status-bar-label-pane(*\text{status});}
\]

See also

- `<status-bar>`
- `status-bar-progress-bar`

**status-bar-progress-bar**

Generic function

Returns the progress bar for the specified status bar.

**Signature**

\[
\text{status-bar-progress-bar玩家朋友 status-bar => progress-bar}
\]

**Parameters**

- `status-bar` – An instance of type `<status-bar>`.

**Values**

- `progress-bar` – An instance of type `false-or(progress-bar)`.

**Discussion**

Returns the progress bar for `status-bar`, if there is one.

See also
Open Abstract Instantiable Class

The class of tab controls.

Superclasses <value-gadget>

Init-Keywords

- **pages** – An instance of type `limited(<sequence>, of: <page>)`.
- **current-page** – An instance of type `false-or(<sheet>)`.
- **key-press-callback** – An instance of type `false-or(<frames.htm#40934>, <function>)`.

Discussion

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.

The `pages` 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 `<page>`.

The `current-page` init-keyword specifies which tab is visible when the tab control is first displayed.

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 tab control, a key-press callback might be used as a quick way to display each page in the tab control. See `gadget-key-press-callback`, for a fuller description of key-press callbacks.

The `gadget-id` of a tab control is particularly useful, because it is returned by `gadget-value`.

Internally, this class maps into the Windows tab control.

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.

```plaintext
contain(make(<tab-control>),
    pages:
        vector(make(<tab-control-page>),
            label: "First",
            child: make(<push-button>),
```
DUIM Reference Documentation, Release 1.0

```lisp
(make (tab-control-page,
    label: "One")),
(make (tab-control-page,
    label: "Second",
    child: make (list-box,
        items: #(1, 2, 3)
    )),
  ) ) ));
```

See also

- `<page>`

### tab-control-current-page Generic function

Returns the current visible page of the specified tab control.

**Signature**  
tab-control-current-page `tab-control` => `visible-page`

**Parameters**

- `tab-control` – An instance of type `<tab-control>`.

**Values**

- `visible-page` – An instance of type `<page>`.

**Discussion**  
Returns the current visible page of `tab-control`.

**Example**  
The following example creates a tab control that has two pages.

```lisp
*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:

```lisp
tab-control-current-page(*tab*);
```

See also

- `<page>`
- `<tab-control>`
- `tab-control-current-page-setter`
- `tab-control-pages`

### tab-control-current-page-setter Generic function

Sets the current visible page of the specified tab control.

**Signature**  
tab-control-current-page-setter `visible-page tab-control` => `visible-page`
Parameters

• **visible-page** – An instance of type `<page>`.
• **tab-control** – An instance of type `<tab-control>`.

Values

• **visible-page** – An instance of type `<page>`.

Discussion Sets the current visible page of `tab-control`.

Example The following example creates a tab control that has two pages.

```duim
*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:

```duim
*page* := tab-control-current-page(*tab*);
```

Next, change the current page of the tab control by clicking on the tab for the hidden page. The, set the current page to be the original current page as follows:

```duim
tab-control-current-page(*tab*) := *page*;
```

See also

• `<page>`
• `<tab-control>`
• `tab-control-current-page`

**tab-control-labels** Generic function

Returns the tab labels of the specified pane.

**Signature** `tab-control-labels tab-control => labels`

**Parameters**

• **tab-control** – An instance of type `<tab-control>`.

**Values**

• **labels** – An instance of type `limited(<sequence>, of: <label>)`.

**Discussion** Returns the tab labels of `tab-control`, as a sequence. Each element in `labels` is an instance of `<label>`.

**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**
- `<tab-control>`
- `tab-control-pages`

**<tab-control-page>** Open Abstract Instantiable Class

The class that represents a page in a tab control.

**Superclasses** `<page>`

**Discussion**

The class that represents a page in a tab control.

![Fig. 9.21: A tab control page](image)

**See also**
- `<page>`
- `<tab-control>`
- `<frames.htm#93333>`
- `<frames.htm#87607>`

**tab-control-pages** Generic function

Returns the tab pages of the specified pane.

**Signature** `tab-control-pages tab-control => pages`

**Parameters**
- `tab-control` – An instance of type `<tab-control>`. 
Values

- **pages** – An instance of type limited(<sequence>, of: <page>). Default value: #[].

Discussion Returns the tab pages of pane.

Example

Given the tab control created by the code below:

```duim
*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>
- <tab-control>
- tab-control-current-page
- tab-control-labels
- tab-control-pages-setter

**tab-control-pages-setter** Generic function

Sets the tab pages of the specified tab control.

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

**Parameters**

- **pages** – An instance of type limited(<sequence>, of: <page>).
- **tab-control** – An instance of <tab-control>.
- **page** – An instance of <page>.

**Values**

- **pages** – An instance of type limited(<sequence>, of: <page>).

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

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

See also
- `<page>`
- `<tab-control>`
- `tab-control-pages`

## <table-column> Sealed Class

The class of columns in table controls.

### Superclasses
 `<object>`

### Init-Keywords

- **heading** – An instance of type `<string>`. 
- **width** – An instance of type `<integer>`. Default value: 100. 
- **alignment** – An instance of type one-of(`"left"`, `"right"`, `"center"`). Default value: `"left"`. 
- **generator** – An instance of type `<function>`. 
- **callback** – An instance of type false-or(`<function>`). Default value: `#f`.

### Discussion

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.

See also
- `<table-control>`

## <table-control> Open Abstract Instantiable Class

The class of table controls.

### Superclasses
 `<collection-gadget>` `<action-gadget>`

### Init-Keywords

- **headings** – An instance of type limited(<sequence>, of: `<string>`). 
- **generators** – An instance of type limited(<sequence>, of: `<function>`). 
- **view** – An instance of type `<table-control-view>`. Default value: `"table"`. 
- **borders** – An instance of type one-of(`#f`, `"none"`, `"flat"`, `"sunken"`, `"raised"`, `"ridge"`, `"groove"`, `"input"`, `"output"`). Default value: `#f`. 

368 Chapter 9. DUIM-Gadgets Library
• **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(<frames.htm#40934>, <function>)`.

• **widths** – An instance of type `limited(<sequence>, of: <integer>)`.

**Discussion**

The class of table controls.

[Image: Diagram of a table control]

The `view`: `init-keyword` can be used to specify how the items in the table control are displayed. See `<table-control-view>`, 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` defines 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`, 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 ‘Defining column headings and contents in table controls’.

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.
Operations

- add-column
- remove-column
- table-control-view
- table-control-view-setter

See also

- <table-column>
- <table-control-view>

<table-control-view> Type

The type of possible views for a table control

Equivalent one-of(#"table", #"small-icon", #"large-icon", #"list")

Discussion

This type represents the acceptable values for the view arguments to operators of <table-control>.

There are four possible values, corresponding to the view options that will be familiar to most users of GUI-based operating systems:

#"small-icon" Displays each item in the table with a small icon to the left of the item. Items are arranged horizontally.

#"large-icon" Displays each item in the table with a large icon to the left of the item. Items are arranged horizontally.

#"list" Displays each item in the table with a small icon to the left of the item. Items are arranged vertically in one column.

#"table" 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.

Fig. 9.22: Defining column headings and contents in table controls
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

• <list-control-view>
• <table-control>
• table-control-view

table-control-view Generic function

Returns the current view of the specified table control.

Signature  
table-control-view  

Parameters

• table-control  –  An instance of type <table-control>.

Values

• view  –  An instance of type <table-control-view>.

Discussion  Returns the current view of table-control. The available views are described in the entry for <table-control-view>.

See also

• <table-control-view>
• table-control-view-setter

table-control-view-setter Generic function

Sets the current view of the specified table control.

Signature  
table-control-view-setter  

Parameters

• view  –  An instance of type <table-control-view>.
• table-control  –  An instance of type <table-control>.

Values

• view  –  An instance of type <table-control-view>.

Discussion  
Sets the current view of table-control.

The view argument is used to specify the way in which the items in the table control are displayed.

See also

• <table-control-view>
• table-control-view

<table-item>  Open Abstract Instantiable Class

The class that represents an item in a table control.

Superclasses <object>

Init-Keywords

9.11. DUIM-Gadgets Module
• **object** – An instance of type `<object>`.

**Discussion**

The class that represents an item in a table control.

The **object**: init-keyword describes the object that an instance of table item represents.

**See also**

- `add-item`
- `find-item`
- `make-item`
- `remove-item`
- `<table-control>`

**<text-editor> Open Abstract Instantiable Class**

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".

**Discussion**

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.

**Example** To constrain the number of lines and columns when an editor is first displayed:

```lisp
*editor* := contain(make(<text-editor>,
lines: 20, columns: 80));
```

To make a text editor that is fixed at 10 lines high:

```lisp
make(<text-editor>, lines: 10, fixed-height?: #t);
```

**See also**

- `<text-field>`
The class of single line text fields.

**Superclasses** `<text-gadget>`

**Init-Keywords**

- **x-alignment** – An instance of type one-of(“left”, “right”, “center”). Default value: “left”.
- **case** – An instance of type one-of(#f, “lower”, “upper”). Default value: #f.
- **auto-scroll?** – An instance of type `<boolean>`. Default value: #f.

**Discussion**

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.

**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>`

**<text-gadget> Open Abstract Class**

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>)`.

**Discussion**

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 `<symbol>`. The string entered in the text gadget is parsed, and converted 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-changing-callback:` 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**

```scheme
contain(make(<text-field>, value-type: <integer>
            text: "1234"));
```

**See also**

• `<combo-box>`

• `gadget-value-type`

• `<password-field>`

• `<text-editor>`

• `<text-field>`

**<tool-bar> Open Abstract Instantiable Class**

The class of tool bars.

**Superclasses** `<gadget>` `<multiple-child-composite-pane>`

**Init-Keywords**
• **update-callback** – An instance of type `<function>`.

**Discussion**

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.

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**

- `<frames.htm#88622>`
- `<frames.htm#56600>`

**See also**

- `<button-box>`
- `<status-bar>`

**<tree-control>** Open Abstract Instantiable Class

The class of tree controls.

**Superclasses** `<collection-gadget>`

**Init-Keywords**

- `children-generator` – An instance of type `<function>`.
- `children-predicate` – An instance of type `<function>`.
- `icon-function` – An instance of type `<function>`.
- `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 `<function>`.
- `key-press-callback` – An instance of type `false-or(<frames.htm#40934>, <function>)`.
- `roots` – An instance of type `<sequence>`. Default value: #[].
Discussion

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`, 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

• 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

Example

```scheme
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
• find-node
• make-node
• remove-node

`tree-control-children-predicate` Generic function
Returns the children predicate function of the specified tree control.

Signature `tree-control-children-predicate tree-control => children-predicate`

Parameters

• `tree-control` – An instance of type `<tree-control>`.

Values

• `children-predicate` – An instance of type `<function>`.

Discussion Returns the children predicate function of `tree-control`.

See also

• `<tree-control>`
• `tree-control-children-predicate-setter`

`tree-control-children-predicate-setter` Generic function
Sets the children predicate function of the specified tree control.

Signature `tree-control-children-predicate-setter children-predicate tree-control => children-predicate`

Parameters

• `children-predicate` – An instance of type `<function>`.
• `tree-control` – An instance of type `<tree-control>`.
Values

- **children-predicate** – An instance of type `<function>`.

Discussion Sets the children predicate function of `tree-control`.

See also

- `<tree-control>`
- `tree-control-children-predicate`
- `tree-control-children-generator-setter`

`tree-control-children-generator` Generic function

Returns the function that generates the children of the specified tree control.

**Signature** `tree-control-children-generator tree-control => children-generator`

**Parameters**

- **tree-control** – An instance of type `<tree-control>`.

**Values**

- **children-generator** – An instance of type `<function>`.

Discussion Returns 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>`
- `tree-control-children-predicate`
- `tree-control-children-generator-setter`

`tree-control-children-generator-setter` Generic function

Sets the function that generates the children of the specified tree control.

**Signature** `tree-control-children-generator-setter children-generator tree-control * => *children-generator`

**Parameters**

- **children-generator** – An instance of type `<function>`.
- **tree-control** – An instance of type `<tree-control>`.

**Values**

- **children-generator** – An instance of type `<function>`.

Discussion 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>`
- `tree-control-children-predicate-setter`
- `tree-control-children-generator`

`tree-control-icon-function` Generic function

Returns the icon function for the specified list control.

**Signature** `tree-control-icon-function tree-control => icon-function`
Parameters

- **tree-control** – An instance of `<tree-control>`.

Values

- **icon-function** – An instance of type `<function>`.

Discussion

Returns the icon function for `tree-control`. 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 `<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.

Note that, unlike list controls, the icon function for a tree control cannot be changed once the list control has been created.

See also

- `list-control-icon-function`
- `<tree-control>`

**tree-control-initial-depth** Generic function

Returns the initial depth of the specified tree control.

**Signature**

```
tree-control-initial-depth tree-control => initial-depth
```

**Parameters**

- **tree-control** – An instance of type `<tree-control>`.

**Values**

- **initial-depth** – An instance of type `<integer>`.

**Discussion**

Returns the initial depth of `tree-control`. 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

- `<tree-control>`
- `tree-control-initial-depth-setter`

**tree-control-initial-depth-setter** Generic function

Sets the initial depth of the specified tree control.

**Signature**

```
tree-control-initial-depth initial-depth tree-control => initial-depth
```

**Parameters**

- **initial-depth** – An instance of type `<integer>`.
- **tree-control** – An instance of type `<tree-control>`.

**Values**

- **initial-depth** – An instance of type `<integer>`.
**Discussion** Sets the initial depth of `tree-control`. 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**
- `<tree-control`
- `tree-control-initial-depth`

**tree-control-roots** Generic function

Returns the roots of the specified tree control.

**Signature**

```
tree-control-roots tree => roots
```

**Parameters**

- `tree` – An instance of type `<tree-control>`.

**Values**

- `roots` – An instance of type `<sequence>`.

**Discussion** 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>`
- `tree-control-roots-setter`

**tree-control-roots-setter** Generic function

Sets the roots of the specified tree control.

**Signature**

```
tree-control-roots-setter roots tree #key frame-manager => roots
```

**Parameters**

- `roots` An instance of type `<sequence>`.
- `tree` An instance of type `<tree-control>`.
- `frame-manager` An instance of type `<frame-manager>`.

**Value**

- `roots` An instance of type `<sequence>`.

**Discussion** 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>
  • tree-control-roots

<tree-node> Open Abstract Instantiable Class

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>.

Discussion

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

  • :gf:contract-node’
  • :gf:expand-node’
  • :gf:node-children’
  • :gf:node-expanded?’
  • :gf:node-parents’

See also
  • <tree-control>
**update-gadget** Generic function

Forces the specified gadget to be redrawn.

**Signature**

\[ \text{update-gadget gadget => ()} \]

**Parameters**

- **gadget** – An instance of type `<gadget>`.

**Discussion**

Forces `gadget` 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**

The class of gadgets that can have values.

**Superclasses** `<gadget>`

**Init-Keywords**

- **value** – An instance of type `<object>`.
- **value-changed-callback** – An instance of type `false-or(<frames.htm#40934>, <function>)`.

**Discussion**

The class of gadgets that can take values.

The `value: 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 `value:` is specified.

The `value-changed-callback: 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.

**Operations**

- `gadget-value`
- `gadget-value-changed-callback`
- `gadget-value-changed-callback-setter`
- `gadget-value-setter`
- `gadget-value-type`

**See also**

- `gadget-value`
- `gadget-value-changed-callback`

**<value-range-gadget> Open Abstract Class**

The class of all value gadgets with ranges.

**Superclasses** `<value-gadget>`

**Init-Keywords**

- **value-range** – An instance of type `<range>`. Default value: `range(from: 0, to: 100)`.

**Discussion**

The class of all value gadgets with ranges. You should not create a direct instance of this class.
The value-range: 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

- gadget-value-range
- gadget-value-range-setter

Example

contain(make(<slider>,
    value-range:
        range(from: -20, to: 20, by: 5)));

See also

- <progress-bar>
- <scroll-bar>
- <slider>
- <value-gadget>

<viewport> Open Abstract Instantiable Class

The class of viewports.

Superclasses <gadget> <single-child-composite-pane>

Init-Keywords

- horizontal-scroll-bar – An instance of type false-or(<scroll-bar>). Default value: #f.
- vertical-scroll-bar – An instance of type false-or(<scroll-bar>). Default value: #f.

Discussion

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
- sheet-viewport-region
- viewport?
- viewport-region

viewport? Generic function
Returns true if the specified object is a viewport.

Signature  viewport? object => viewport?

Parameters

- object – An instance of type <object>.

Values

- viewport? – An instance of type <boolean>.

Discussion  Returns true if object is a viewport.

Example  To test whether the gadget *gadget* is a viewport:

```lisp
viewport?(*gadget*);
```

See also

- <viewport>
- <button-box>
- <border>

viewport-region Generic function
Returns the region for the specified viewport.

Signature  viewport-region viewport => region

Parameters

- viewport – An instance of type <viewport>.

Values

- region – An instance of type <region>.

Discussion  Returns the region for viewport.

Example  To return the region for a viewport *viewer*:

```lisp
viewport-region(*viewer*);
```

See also

- <viewport>

with-border Statement Macro
Creates the specified sheet and places a border around it.

Macro Call

```lisp
with-border ([*options* ] ) {*pane* } end
```
Parameters

- **options** – Dylan arguments *bnf*.
- **pane** – A Dylan expression *bnf*.

Discussion

Creates *pane* with a border around 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 `<border>`. If no options are specified, then the default border is used.

The pane is an expression whose return value is the sheet around which a border should be placed.

Example

To create a button in a border:

```dylan
contain(with-border (type: #"raised")
  make(<button>,
      label: "Hello") end);
```

See also

- `<border>`
- `labelling`
- `with-spacing`

**with-spacing** Statement Macro

Creates the specified sheet and places spacing around it.

Macro Call

```
with-spacing ([*options* ]) (*pane* ) end
```

Parameters

- **options** – Dylan arguments *bnf*.
- **pane** – A Dylan expression *bnf*.

Discussion

Creates *pane* with spacing around 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 `<spacing>`. If no options are specified, then the default spacing is used.

The pane is an expression whose return value is the sheet around which spacing should be placed.

Example

To create a button with spacing:

```dylan
contain(with-spacing (thickness: 10)
  (vertically () make(<button>,
      label: "Hello")
  make(<button>,
      label: "World")
  end)
end);
```

See also

- `<null-pane>`
- \texttt{<spacing>}
- \texttt{with-border}
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. *DUIM-Frames Module* 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 *Subclasses of <frame-event>*.

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.

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.

- `<object>`
  - `<gadget>`
  - `<page>`
    * See *Subclasses of <page>*
  - `<frame>`
    * See *Subclasses of <frame>*
  - `<event>`
    * `<frame-event>`
The <frame> class represents the base class for all types of frame. An introduction to the subclasses available is given in Subclasses of <frame>.

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 Subclasses of <frame-event> for an introduction to these subclasses. See the 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.

Subclasses of <frame>

A number of subclasses of <frame> are provided to allow you to create a variety of common types of frame.

• <frame>
  • <simple-frame>
  • <dialog-frame>
  • <property-frame>
  • <wizard-frame>

<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 sequential steps, filling out any information requested and using Next and Back buttons to navigate to the next or previous steps in the process.
Subclasses of <frame-event>

The <frame-event> class provides a number of subclasses that describe various events that can occur in frames.

- <frame-event>
  - <frame-created-event>
  - <frame-destroyed-event>
  - <frame-mapped-event>
  - <frame-unmapped-event>
  - <frame-exit-event>
  - <frame-exited-event>
  - <application-exited-event>

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).

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 Subclasses of <page> for an introduction to these classes.

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.

**DUIM-Frames Module**

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

= (command) Method

Returns true if the specified commands are the same.

**Signature**

\[
\text{equal?} \leftarrow \text{command1} \text{ command2} \\
\]

**Parameters**

- \(\text{command1}\) – An instance of type `<command>`.
- \(\text{command2}\) – An instance of type `<command>`.

**Values**

- \(\text{equal?}\) – An instance of type `<boolean>`.

**Discussion**

Returns true if \(\text{command1}\) and \(\text{command2}\) are the same.

**add-command** Generic function

Adds a command to the specified command table.

**Signature**

\[
\text{add-command} \leftarrow \text{command-table} \text{ command} \text{ name} \text{ menu} \text{ image} \text{ accelerator} \text{ mnemonic} \text{ error?} \Rightarrow () \\
\]

**Parameters**

- \(\text{command-table}\) – An instance of type `<command-table>`.
- \(\text{command}\) – An instance of type `type-union(<command>, <function>)`.
- \(\text{name}\) (#key) – An instance of type `false-or(<string>)`.
- \(\text{menu}\) (#key) – An instance of type `false-or(<menu>)`.
- \(\text{image}\) (#key) – An instance of type `false-or(<image>)`.
- \(\text{accelerator}\) (#key) – An instance of type `false-or(<gesture>)`.
- \(\text{mnemonic}\) (#key) – An instance of type `false-or(<gesture>)`.
- \(\text{error?}\) (#key) – An instance of type `<boolean>`.
  Default value: `#t`.

**Discussion**

You can supply a keyboard accelerator or a mnemonic using the `accelerator` and `mnemonic` arguments respectively.

Adds \text{command} to \text{command-table}.

The argument \text{name} is the command-line name for the command.

- When \text{name} is `#f`, the command is not available via command-line interactions.
- When \text{name} is a string, that string is the command-line name for the command.

For the purposes of command-line name lookup, the character case of \text{name} is ignored.

The argument \text{menu} is a menu for \text{command}.

- When \text{menu} is `#f`, \text{command} is not available via menus.
• When menu is a string, the string is used as the menu name.
• 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

add-command-table-menu-item Generic function
Adds a menu item to the specified command table.

Signature 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 => menu-item

Parameters
• command-table – An instance of type <command-table>.
• string – An instance of type false-or(<string>).
• type – An instance of type one-of(#"command", #"function", #"menu", #"divider").
• value – An instance of type <object>.
• documentation (#key) – An instance of type <string>.
• after (#key) – An instance of type one-of(#"start", #"end", #"sort"), or an instance of <string>. Default value: #"end".
• accelerator (#key) – An instance of type false-or(<gesture>).
• mnemonic (#key) – An instance of type false-or(<gesture>).
• text-style (#key) – An instance of type <text-style>.
• error? (#key) – An instance of type <boolean>. Default value: #t.
• items (#key) – An instance of type limited(<sequence>, of: ).
• label-key (#key) – An instance of type <function>.
• value-key (#key) – An instance of type <function>.
• test (#key) – An instance of type <function>.
• callback (#key) – An instance of type <function>.

Values

• menu-item – An instance of type <command-table-menu-item>.

Discussion

Adds a command menu item to the menu in command-table. The string argument is the name of the command menu item; its character case is ignored. The type of the item is either "command", "function", "menu", or "divider".

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:

- 
  • 
  
  • 

A string naming an existing entry

- 
  • 

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 

When type is 

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 

error is signalled. When the item is already present in the command table and error? is , 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>
- remove-command-table-menu-item

<application-exited-event> Instantiable Sealed Class

The class of events signalled when an application exits.

Superclasses

Discussion

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.

See also

- exit-frame
- <frame-exited-event>

apply-in-frame Generic function

Applies the specified function to the given arguments in the main thread of the frame.

Signature

Parameters

- frame – An instance of type <frame>.
- function – An instance of type <function>.
- arg – An instance of type <object>.
- args (#rest) – Instances of type <object>.
Discussion  Applies function to the given arguments in the main thread of frame. You must supply at least one argument (arg), though you can optionally supply as many additional arguments as you like.

See also
  • call-in-frame

call-in-frame Generic function
  Calls the specified function with the given arguments in the main thread of the frame.

Signature  call-in-frame frame function #rest args => ()

Parameters
  • frame – An instance of type <frame>.
  • function – An instance of type <function>.
  • args (#rest) – Instances of type <object>.

Discussion  Calls function with the given arguments in the main thread of frame.

See also
  • apply-in-frame

cancel-dialog Generic function
  Cancels the specified dialog.

Signature  cancel-dialog dialog #key destroy? => ()

Parameters
  • dialog – An instance of type <dialog-frame>.
  • destroy? – An instance of type <boolean>. Default value: #t.

Discussion  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);
define variable *dialog* = make(<dialog-frame>, exit-button?: #f, cancel-button?: #f, layout: vertically () make(<label>, label: "Simple dialog")); horizontally () *yes-button*;
```
clear-progress-note Generic function
Clears the specified progress note.

Signature  
clear-progress-note framem progress-note => ()

Parameters
• framem – An instance of type <frame-manager>.
• progress-note – An instance of type <progress-note>.

Discussion  
Clears the specified progress note.

<command> Open Abstract Instantiable Class
The class of commands.

Superclasses  <object>

Init-Keywords
• function – An instance of type <function>.
• arguments – An instance of type <sequence>. Default value: #[].

Discussion  
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 execute-command.

The function: init-keyword is the command function that is called by the command object. A command function is rather like a callback to a <command> object: a command can be executed via execute-command, which then invokes the command function. Command functions take at least one argument: a <frame> object.

The arguments: init-keyword are the arguments passed to the command function.

Operations
• =
• add-command
• command-arguments
• command-enabled?
• command-enabled?-setter
• command-function
• command-undoable?
• dialog-cancel-callback-setter
• dialog-exit-callback-setter
• execute-command
• gadget-command
• gadget-command-setter
• gadget-key-press-callback-setter
• redo-command
• remove-command
• undo-command

See also
• command?
• command-arguments
• command-function
• execute-command
• <simple-command>

**command?** Generic function
Returns true if the specified object is a command.

**Signature**  
command? object => command?

**Parameters**
• object – An instance of type <object>.

**Values**
• command? – An instance of type <boolean>.

**Discussion**  
Returns true if object is an instance of <command>.

**See also**
• <command>

**command-arguments** Generic function
Returns the arguments to the specified command.

**Signature**  
command-arguments command => arguments

**Parameters**
• command – An instance of type <command>.

**Values**
• arguments – An instance of type <sequence>.

**Discussion**  
Returns the arguments to command.

**See also**
• <command>
command-enabled? Generic function
Returns true if the specified command is enabled.

**Signature**  
`command-enabled? command frame #key => enabled?`

**Parameters**
- `command` – An instance of type `type-union(<command>, <command-table>)`.
- `frame` – An instance of type `<frame>`.

**Values**
- `enabled?` – An instance of type `<boolean>`.

**Discussion**  
Returns true if `command in frame` is enabled.

**See also**
- `<command>`
- `command-enabled?-setter`

command-enabled?-setter Generic function
Enables or disables the specified command.

**Signature**  
`command-enabled?-setter enabled? command frame => enabled?`

**Parameters**
- `enabled?` – An instance of type `<boolean>`.
- `command` – An instance of type `type-union(<command>, <command-table>)`.
- `frame` – An instance of type `<frame>`.

**Values**
- `enabled?` – An instance of type `<boolean>`.

**Discussion**
Enables or disables `command in frame`. If `enabled?` is true, then `command` 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.

This function is useful when manipulating the disabled commands in `frame`. For example, it is common to disable the `Save` menu command immediately after saving a file, enabling it again only when the file has been modified.

**See also**
- `command-enabled?`

command-function Generic function
Returns the function associated with the specified command.

**Signature**  
`command-function command => function`

**Parameters**
- `command` – An instance of type `<command>`.

**Values**
- `function` – An instance of type `<function>`.
**Discussion**  Returns the function associated with *command*. A command function is the function that is called by a `<command>` 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 `<frame>` object.

**See also**
- `<command>`
- `execute-command`

**<command-table> Open Abstract Instantiable Class**

The class of command tables.

**Superclasses** `<object>`

**Init-Keywords**
- `name` – An instance of type `<object>`. Required.
- `inherit-from` – An instance of type `limited(<sequence>, of: <command-table>). Required.
- `resource-id` – An instance of type `false-or(<object>)`. Default value: #f.

**Discussion**

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.

The `name`: init-keyword is a symbol that names the current command table.

The `inherit-from`: 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 `*global-command-table*`, and can also inherit the behavior specified by `*user-command-table*`.

You do not normally need to specify a unique `resource-id`: yourself. As with most other DUIM classes, the `name`: init-keyword serves as a sufficient unique identifier.

**Operations**
- `add-command`
- `add-command-table-menu-item`
- `command-table-accelerators`
- `command-table-commands`
- `command-table-menu`
- `command-table-name`
- `frame-command-table-setter`
- `make(<frame>)`
- `make-menu-from-command-table`
- `make-menus-from-command-table`
- `remove-command`
- `remove-command-table`
- `remove-command-table-menu-item`
Example

```
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*;
```

See also

• *global-command-table*
• *user-command-table*

command-table? Generic function

Returns true if the specified object is a command table.

Signature command-table? object => command-table?

Parameters

• object – An instance of type <object>.

Values

• command-table? – An instance of type <boolean>.

Discussion Returns true if object is a command table.

See also

• <command-table>

command-table-accelerators Generic function

Returns the keyboard accelerators for the specified command table.

Signature command-table-accelerators command-table => accelerators

Parameters

• command-table – An instance of type <command-table>.

Values

• accelerators – An instance of type limited(<sequence>, of: <gesture>).

Discussion Returns the keyboard accelerators for command-table.

See also

• command-table-commands

command-table-commands Generic function

Returns the commands for the specified command table.

Signature command-table-commands command-table => commands

Parameters

• command-table – An instance of type <command-table>.
Values

- **commands** – An instance of type `limited(<sequence>, of: <command>).`

Discussion  Returns the commands defined for `command-table`.

See also

- `command-table-accelerators`
- `command-table-menu`

**command-table-menu** Generic function

Returns the menu items in the specified command table.

**Signature**  `command-table-menu command-table => menu-items`

**Parameters**

- `command-table` – An instance of type `<command-table>`.

**Values**

- `menu-items` – An instance of type `<stretchy-vector>`.

Discussion  Returns the menu items in `command-table`.

See also

- `command-table-commands`
- `command-table-name`

**<command-table-menu-item>** Instantiable Sealed Class

The class of menu items in command tables.

**Superclasses**  `<object>`

**Init-Keywords**

- `name` – An instance of type `false-or(<string>).` Default value: #f.
- `image` – An instance of type `false-or(type-union(<string>, <image>)).` Default value: #f
- `type` – An instance of type `one-of(#"command", #"function", #"menu", #"divider").`
- `value` – An instance of type `<object>`. Default value: #f.
- `options` – An instance of type `<sequence>`. Default value: #().
- `accelerator` – An instance of type `false-or(<gesture>).` Default value: #f.
- `mnemonic` – An instance of type `false-or(<gesture>).` Default value: #f.

Discussion  The class of menu items in command tables. This class models menu items, tool bar items, accelerators, and mnemonics for a command table entry.

The **type**: init-keyword denotes what type of menu item has been created. This is either #"command", #"function", #"menu", or #"divider".

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 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

• add-command-table-menu-item
• menu-item-accelerator
• menu-item-mnemonic
• menu-item-name
• menu-item-options
• menu-item-type
• menu-item-value

See also

• add-command-table-menu-item

command-table-name Generic function

Returns the name of the specified command table.

Signature  command-table-name command-table => name

Parameters

• command-table – An instance of type <command-table>.

Values

• name – An instance of type <object>.

Discussion  Returns the name of command-table, as defined by the name: init-keyword for <command-table>.

See also

• <command-table>
• command-table-menu
command-undoable? Generic function

Returns true if the specified command is undoable.

Signature  command-undoable? command => undoable?

Parameters

- command – An instance of type <command>.
- undoable? – An instance of type <boolean>.

Discussion  Returns true if command is undoable, that is, there is a specified command that the user can choose (for instance, by choosing Edit > Undo) that will reverse the effects of command.

See also

- undo-command

complete-from-generator Generic function

Completes a string based on a generated list of completions.

Signature  complete-from-generator string generator delimiters #key action predicate => string success object nmatches completions

Parameters

- string – An instance of type <string>.
- generator – An instance of type <function>.
- delimiters – An instance of type limited(<sequence>, of: <character>).
- action – An instance of type one-of(#"complete", #"complete-limited", #"complete-maximal", #"completions", #"apropos-completions"). Default value #"complete".
- predicate – An instance of type false-or(<function>). Default value #f.

Values

- string – An instance of type false-or(<string>).
- success – An instance of type <boolean>.
- object – An instance of type <object>.
- nmatches – An instance of type <integer>.
- completions – An instance of type <sequence>.

Discussion  Completes string chunk-wise against a list of possibilities derived from generator, using the specified delimiters to break both string and the generated possibilities into chunks. This function is identical to complete-from-sequence, except that the list of possibilities is derived from the generator function, rather than passed explicitly. The generator 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.

The argument predicate (if supplied) is applied to filter out unwanted objects.

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 action is #"completions") a sequence of possible completions.
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**

```scheme
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**

**complete-from-sequence** Generic function

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
```

**Parameters**

- **string** – An instance of type `<string>`.
- **possibilities** – An instance of type `limited(<sequence>, of: <string>)`.
- **delimiters** – An instance of type `limited(<sequence>, of: <character>)`.
- **action (#key)** – An instance of type `one-of(#"complete", #"complete-limited", #"complete-maximal", #"completions", #"apropos-completions")`. Default value #"complete".
- **predicate (#key)** – An instance of type `false-or(<function>)`. Default value #f.
- **label-key (#key)** – An instance of type `<function>`. Default value `first`.
- **value-key (#key)** – An instance of type `<function>`. Default value `second`.

**Values**

- **string** – An instance of type `false-or(<string>)`.
- **success** – An instance of type `<boolean>`.
- **object** – An instance of type `<object>`.
- **nmatches** – An instance of type `<integer>`.
- **completions** An instance of type `<sequence>`.
Discussion

Completes string chunk-wise against the list of possibilities, using the specified delimiters to break both string and the strings in possibilities into chunks.

The label-key and value-key arguments are used to extract the completion string and object from the entries in possibilities, and predicate (if supplied) is applied to filter out unwanted objects.

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 action is #"completions") a sequence of possible completions.

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

```scheme
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

compute-next-page Generic function

Returns the next page in the specified wizard frame.

Signature  compute-next-page dialog => next-page

Parameters

• dialog – An instance of type <wizard-frame>.

Next-page  An instance of type false-or(<sheet>).

Discussion  Returns the next page in dialog, which must be a wizard.

See also

• compute-previous-page

• <wizard-frame>

compute-previous-page Generic function

Returns the previous page in the specified wizard frame.

Signature  compute-previous-page dialog => prev-page

Parameters
• **dialog** – An instance of type `<wizard-frame>`.

**Values**

• **prev-page** – An instance of type `false-or(<sheet>)`.

**Discussion** Returns the previous page in `dialog`, which must be a wizard.

**See also**

• `compute-next-page`

• `<wizard-frame>`

---

**contain**

Generic function

Creates and returns a frame containing the specified object.

**Signature**

```
contain object #rest initargs #key own-thread? #all-keys => sheet frame
```

**Parameters**

• **object** – An instance of type `type-union(<sheet>, <class>, <frame>)`.

• **initargs** – Instances of type `<object>`.

• **own-thread? (#key)** – An instance of type `<boolean>`.

**Values**

• **sheet** – An instance of type `<sheet>`.

• **frame** – An instance of type `<frame>`.

**Discussion**

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 not 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.

**current-frame Function**

Returns the current frame

**Signature** current-frame => frame

**Values**

- **frame** – An instance of type <frame>

**Discussion** Returns the current frame.

**define command-table Defining Macro**

Defines a new class of command table with the specified name and properties.

**Macro Call**

```
define command-table *name* (*supers*, *) (*options* ) end
```

**Parameters**

- **name** – A Dylan name bnf.
- **supers** – A Dylan name bnf.
- **options** – A Dylan body bnf.

**Discussion**

Defines a new class of command table with the specified name and properties. This macro is equivalent to define class, but with additional options.

The supers 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 supers should have the value *global-command-table*.

Each one of the options 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:

```
menu-item *menu-item-descriptor* ;
include *command-table-name* ;
```
To add a menu item or menu to a command table, include an option of the following form:

```plaintext
menu-item *label* = *command-function* #key *accelerator documentation*
```

- **label** An instance of `<string>`. This is the label that appears in the menu.
- **command-function** An instance of `type-union(<command>, <command-table>, <function>)`. 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.
- **accelerator** An instance of `false-or(<gesture>)`. Default value: `#f`. This defines a keyboard accelerator that can be used to invoke `command-function` in preference to the menu item itself.
- **documentation** An instance of `false-or(<string>)`. Default value: `#f`. 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:

```plaintext
separator;
```

To include another command table in the current table, include the following option at the point you want the command table to appear:

```plaintext
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.

```plaintext
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>),
```
DUIM Reference Documentation, Release 1.0

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*

define frame Defining Macro
Defines a new class of frame with the specified properties.

Macro Call

define frame *name* ({*supers* }, *) {*slots-panes-options* } end

Parameters

• name – A Dylan name bnf.
• supers – A Dylan name bnf.
• slots-panes-options – A Dylan body bnf.

Discussion

Defines a new class of frame called *name* with the specified properties. This macro is equivalent to *define class*, but with additional options.

The *supers* 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 <simple-frame> or <dialog-frame>.

The *slots-panes-options* 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:

• A number of slots for defining per-instance values of the frame state.
• A number of named panes. Each pane defines a sheet of some sort.
• A single layout.
• A tool bar.
• A status bar.
• A menu bar.
• A command table.
• A number of sequential pages for inclusion in a multi-page frame such as a wizard or property dialog.
**Note:** 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.

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:

```plaintext
*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:
  - `#f` The frame has no menu bar.
  - `#t`, The menu bar for the frame is defined by the value of the *command-table* 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:
The name, supers, and slot arguments are not evaluated. The values of each of the options are evaluated.

Example

```lisp
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>`
- `<wizard-frame>`

**deiconify-frame**

Generic function

Displays a frame that has previously been iconified on screen.

**Signature**  
deiconify-frame frame => ()

**Parameters**
• **frame** – An instance of type `<frame>`.

Discussion 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.

```lisp
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**
• **exit-frame**
• **frame-icon**
• **iconify-frame**

**destroy-frame** Generic function

Unmaps the specified frame and destroys it.

Signature `destroy-frame frame => ()`

Parameters

• **frame** – An instance of type `<frame>`.

Discussion Unmaps frame from the screen and destroys it. Generally, you should not need to call this function explicitly, since `exit-frame` performs all necessary operations in the correct order, including calling `destroy-frame` if the `destroy?` argument to `exit-frame` is true.

See also

• **deiconify-frame**
• **exit-frame**
• `<frame-destroyed-event>`
• **iconify-frame**
• **lower-frame**
• **raise-frame**

**dialog-apply-button** Generic function

Returns the Apply button in the specified dialog.

Signature `dialog-apply-button dialog => apply-button`

Parameters

• **dialog** – An instance of type `<dialog-frame>`.

Values

• **apply-button** – An instance of type `false-or(<button>)`.

Discussion Returns the Apply button in dialog. 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.
dialog-apply-button-setter Generic function
Specifies the Apply button in the specified dialog.

Signature  dialog-apply-button-setter apply-button dialog => apply-button

Parameters
- apply-button – An instance of type false-or(<button>).
- dialog – An instance of type <dialog-frame>.

Values
- apply-button – An instance of type false-or(<button>).

Discussion  Specifies the Apply button in dialog. 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
- dialog-cancel-button
- dialog-apply-button
- dialog-apply-callback
- dialog-help-button

dialog-apply-callback Generic function
Returns the callback invoked when the Apply button is clicked in the specified dialog.

Signature  dialog-apply-callback dialog => callback

Parameters
- dialog – An instance of type <dialog-frame>.

Values
- callback – An instance of type false-or(<command>, <function>).

Discussion
Returns the callback invoked when the Apply button is clicked in dialog. 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.

Note: If you supply #f as the callback, then the button does not appear.
dialog-back-button Generic function

Returns the Back button in the specified multi-page dialog.

Signature  dialog-back-button dialog => back-button

Parameters

- dialog – An instance of type <dialog-frame>.

Values

- back-button – An instance of type false-or(<button>).

Discussion  Returns the Back button in dialog. 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

- dialog-back-button-setter
- dialog-back-callback
- dialog-exit-button
- dialog-help-button

dialog-back-button-setter Generic function

Specifies the Back button in the specified multi-page dialog.

Signature  dialog-back-button-setter back-button dialog => back-button

Parameters

- back-button – An instance of type <button>.
- dialog – An instance of type <dialog-frame>.

Values

- back-button – An instance of type <button>.

Discussion  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
- dialog-back-callback
- dialog-exit-button-setter
- dialog-help-button

dialog-back-callback Generic function

Returns the callback invoked when the Back button is clicked in the specified multi-page dialog.

Signature  dialog-apply-callback dialog => callback

Parameters

- dialog – An instance of type <dialog-frame>.

Values
• **callback** – An instance of type `false-or(<command>, <function>)`.

**Discussion**

Returns the callback invoked when the Back button is clicked 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.

**Note:** 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**

- `dialog-back-button`
- `dialog-back-button-setter`
- `dialog-exit-callback`
- `dialog-help-button`

**dialog-cancel-button** Generic function

Returns the Cancel button in the specified dialog.

**Signature** `dialog-cancel-button dialog => cancel-button`

**Parameters**

- `dialog` – An instance of type `<dialog-frame>`.

**Values**

- `cancel-button` – An instance of type `false-or(<button>)`.

**Discussion** Returns the Cancel button in `dialog`.

**See also**

- `dialog-cancel-button-setter`
- `dialog-cancel-callback`
- `dialog-exit-button`
- `dialog-help-button`

**dialog-cancel-button-setter** Generic function

Specifies the Cancel button in the specified dialog.

**Signature** `dialog-cancel-button-setter cancel-button dialog => cancel-button`

**Parameters**

- `cancel-button` – An instance of type `<button>`.
- `dialog` – An instance of type `<dialog-frame>`.

**Values**

- `cancel-button` – An instance of type `<button>`.

**Discussion** 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.

```lisp
(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`
- `dialog-cancel-callback`
- `dialog-exit-button-setter`
- `dialog-help-button-setter`

**dialog-cancel-callback** Generic function
Returns the function invoked when the cancel button is clicked in the specified dialog.

**Signature** `dialog-cancel-callback dialog => callback`

**Parameters**
- `dialog` – An instance of type `<dialog-frame>`.

**Values**
- `callback` – An instance of type `false-or(type-union(<command>, <function>))`.

**Discussion** Returns the function invoked when the cancel button is clicked in `dialog`. This defaults to `cancel-dialog`.

See also
- `cancel-dialog`
- `dialog-cancel-button`
- `dialog-cancel-button-setter`
- `dialog-exit-callback`
- `dialog-help-callback`

**dialog-cancel-callback-setter** Generic function
Sets the function invoked when the cancel button is clicked in the specified dialog.

**Signature**
`dialog-cancel-callback-setter callback dialog => callback`
param callback An instance of type false-or(<command>, <function>).
Default value: cancel-dialog.

param dialog An instance of type <dialog-frame>.

type callback An instance of type false-or(<command>, <function>).

Discussion Sets the function invoked when the cancel button is clicked in dialog.

See also
- dialog-cancel-button
- dialog-cancel-button-setter
- dialog-exit-callback
- dialog-help-callback

dialog-current-page Generic function
Returns the current page in the specified multi-page dialog.

Signature dialog-current-page dialog => page

Parameters
- dialog – An instance of type <dialog-frame>.

Values
- page – An instance of type false-or(<page>).

Discussion Returns the current page in dialog.

See also
- dialog-current-page-setter

dialog-current-page-setter Generic function
Sets the current page in the specified multi-page dialog.

Signature dialog-current-page-setter page dialog => page

Parameters
- page – An instance of type <page>.
- dialog – An instance of type <dialog-frame>.

Values
- page – An instance of type <page>.

Discussion Sets the current page in dialog.

See also
- dialog-current-page

dialog-exit-button Generic function
Returns the Exit button in the specified dialog.

Signature dialog-exit-button dialog => exit-button

Parameters
- dialog – An instance of type <dialog-frame>.

Values
• **exit-button** – An instance of type `false-or(<button>)`.

**Discussion** Returns the Exit button in `dialog`. 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-cancel-button`
- `dialog-exit-button-setter`
- `dialog-exit-enabled?`
- `dialog-exit-callback`
- `dialog-help-button`

**dialog-exit-button-setter** Generic function

Specifies the Exit button in the specified dialog.

**Signature**`dialog-exit-button-setter` `exit-button` `dialog` => `exit-button`

**Parameters**

- `exit-button` – An instance of type `<button>`.
- `dialog` – An instance of type `<dialog-frame>`.

**Values**

- `exit-button` – An instance of type `<button>`.

**Discussion** Sets the Exit button in `dialog`. 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.

```Scheme
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`
- `dialog-exit-button`
- `dialog-exit-enabled?`
- `dialog-exit-callback`
• `dialog-help-button-setter`

**dialog-exit-callback** Generic function

Returns the callback invoked when the Exit button is clicked in the specified dialog.

**Signature** `dialog-exit-callback dialog => callback`

**Parameters**

- `dialog` – An instance of type `<dialog-frame>`.

**Values**

- `callback` – An instance of type `false-or(type-union(<command>,<function>))`. Default value: `exit-dialog`.

**Discussion** Returns the callback invoked when the Exit button is clicked in `dialog`. 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-cancel-callback`
- `dialog-exit-button`
- `dialog-exit-button-setter`
- `dialog-exit-callback-setter`
- `dialog-help-callback`

**dialog-exit-callback-setter** Generic function

Sets the callback invoked when the Exit button is clicked in the specified dialog.

**Signature** `dialog-exit-callback callback dialog => callback`

**Parameters**

- `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>))`.

**Discussion** Sets the callback invoked when the Exit button is clicked in `dialog`. 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 `#f` means that no Exit button is displayed at all.

See also

- `dialog-cancel-callback-setter`
- `dialog-exit-button`
- `dialog-exit-button-setter`
dialog-exit-enabled? Generic function
Returns true if the Exit button has been enabled for the specified dialog.

Signature  dialog-exit-enabled? dialog => enabled?

Parameters
  • dialog – An instance of type <dialog-frame>.

Values
  • enabled? – An instance of type <boolean>.

Discussion  Returns true if the Exit button has been enabled for dialog. 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
  • dialog-exit-button-setter
  • dialog-exit-enabled?-setter
  • dialog-exit-callback

dialog-exit-enabled?-setter Generic function
Enables or disables the Exit button for the specified dialog.

Signature  dialog-exit-enabled?-setter enabled? dialog => enabled?

Parameters
  • enabled? – An instance of type <boolean>.
  • dialog – An instance of type <dialog-frame>.

Values
  • enabled? – An instance of type <boolean>.

Discussion  Enables or disables the Exit button for dialog. 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.

```lisp
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`
• `dialog-exit-button-setter`
• `dialog-exit-enabled?`
• `dialog-exit-callback`

**<dialog-frame> Open Abstract Instantiable Class**
The class of dialog frames.

**Superclasses** `<simple-frame>`

**Init-Keywords**

- `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`.  
- `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`.  
- `pages` – An instance of type `false-or(<sequence>)`. Default value: `#f`.  
- `page-changed-callback` – An instance of type `false-or(<function>)`. Default value: `#f`.  

**Discussion**
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.

![Fig. 10.1: A typical dialog](image)

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 \texttt{mode:}
keyword to \texttt{"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 \texttt{init-keywords} \texttt{exit-button:}, and \texttt{cancel-button:} specify the exit and cancel but-
tons 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 \texttt{exit-callback:} and \texttt{cancel-callback:} \texttt{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 spec-
ify an alternative if you wish. If you do not require a Cancel button in your dialog, specify
\texttt{cancel-callback: #f}. Similarly, specify \texttt{exit-callback: #f} if you do not re-
quire 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 \texttt{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 \texttt{exit-enabled?:} \texttt{init-keyword} is used to specify whether the exit button on the dialog
is enabled or not. If \texttt{#f}, then the exit button is displayed on the dialog, but it is grayed out.

The \texttt{help-button:} \texttt{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 \texttt{help-callback:} \texttt{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 \texttt{exit-buttons-position:} \texttt{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 \texttt{pages:} \texttt{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 \texttt{page-changed-callback:} is a callback function that is invoked when a different
page in a multi-page dialog is displayed.

**Operations**

- \texttt{cancel-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.

```scheme
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
- exit-dialog
- <property-frame>
- <simple-frame>
- <wizard-frame>

dialog-help-button Generic function

Returns the Help button in the specified dialog.

Signature  dialog-help-button dialog => help-button

Parameters

- dialog – An instance of type <dialog-frame>.
Values

- **help-button** – An instance of type `false-or(<button>)`.

**Discussion** Returns the Help button in `dialog`. Many dialogs contain a Help button that, when clicked, displays a relevant topic from the online help system for the application.

**See also**

- `dialog-cancel-button`
- `dialog-exit-button`
- `dialog-help-button-setter`
- `dialog-help-callback`

**dialog-help-button-setter** Generic function

Specifies the Help button in the specified dialog.

**Signature** `dialog-help-button-setter help-button dialog => help-button`

**Parameters**

- **help-button** – An instance of type `false-or(<button>)`.
- **dialog** – An instance of type `<dialog-frame>`.

**Values**

- **help-button** – An instance of type `false-or(<button>)`

**Discussion** Specifies the Help button in `dialog`. 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.

```lisp
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
- dialog-exit-button-setter
- dialog-help-button
- dialog-help-callback

**dialog-help-callback** Generic function

Returns the callback invoked when the Help button is clicked in the specified dialog.

**Signature**  
dialog-help-callback [dialog] => help-callback

**Parameters**

- **dialog** – An instance of type `<dialog-frame>`.

**Values**

- **help-callback** – An instance of type false-or(type-union(<command>, <function>)).

**Discussion**

Returns the callback invoked when the Help button is clicked in **dialog**. Many dialogs contain a Help button that, when clicked, displays a relevant topic from the online help system for the application.

**Note:** 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**

- dialog-cancel-callback
- dialog-exit-callback
- dialog-help-button
- dialog-help-button-setter

**dialog-next-button** Generic function

Returns the Next button in the specified multi-page dialog.

**Signature**  
dialog-next-button [dialog] => next-button

**Parameters**

- **dialog** – An instance of type `<dialog-frame>`.

**Values**

- **next-button** – An instance of type false-or(<button>).

**Discussion**

Returns the Next button in **dialog**. 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**

- dialog-back-button
- dialog-exit-button
- dialog-next-button-setter
• **dialog-next-callback**

**dialog-next-button-setter** Generic function

Specifies the Next button in the specified multi-page dialog.

**Signature**  
dialog-next-button-setter next-button dialog => next-button

**Parameters**

- **next-button** – An instance of type false-or(button).
- **dialog** – An instance of type dialog-frame.

**Values**

- **next-button** – An instance of type false-or(button).

**Discussion**

Specifies the Next button in dialog. 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**

- dialog-back-button-setter
- dialog-exit-button
- dialog-next-button
- dialog-next-callback

**dialog-next-callback** Generic function

Returns the callback invoked when the Next button is clicked in the specified multi-page dialog.

**Signature**  
dialog-next-callback dialog => callback

**Parameters**

- **dialog** – An instance of type dialog-frame.

**Values**

- **callback** – An instance of type false-or(type-union(command), function).

**Discussion**

Returns the callback invoked when the Next button is clicked in dialog. 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.

**Note:** 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.

The default value for this callback is move-to-next-page.

**See also**

- dialog-back-button
- dialog-exit-callback
• **dialog-next-button**
• **dialog-next-button-setter**
• **move-to-next-page**

**dialog-next-enabled?** Generic function

Returns true if the Next button has been enabled for the specified multi-page dialog.

**Signature**  
dialog-next-enabled? dialog => enabled?

**Parameters**

- **dialog** – An instance of type `<dialog-frame>`.

**Values**

- **enabled?** – An instance of type `<boolean>`.

**Discussion**

Returns true if the Next button has been enabled for `dialog`. 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**

- `<dialog-frame>`
- **dialog-next-button**
- **dialog-next-button-setter**
- **dialog-next-enabled?-setter**
- **dialog-next-callback**

**dialog-next-enabled?-setter** Generic function

Enables or disables the Next button for the specified multi-page dialog.

**Signature**  
dialog-next-enabled?-setter enabled? dialog => enabled?

**Parameters**

- **enabled?** – An instance of type `<boolean>`.
- **dialog** – An instance of type `<dialog-frame>`.

**Values**

- **enabled?** – An instance of type `<boolean>`.

**Discussion**

Enables or disables the Next button for `dialog`. 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.

**See also**

- **dialog-next-button**
- **dialog-next-button-setter**
dialog-next-page

Generic function

Returns the next page in sequence for the specified multi-page dialog.

**Signature**

dialog-next-page dialog => next-page

**Parameters**

- **dialog** – An instance of type `<dialog-frame>`.  

**Values**

- **next-page** – An instance of type `false-or(<page>)`.  

**Discussion**

Returns 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**

- `dialog-next-button`
- `dialog-next-button-setter`
- `dialog-next-callback`
- `dialog-next-page-setter`
- `dialog-previous-page`

dialog-next-page-setter

Generic function

Specifies the next page in sequence for the specified multi-page dialog.

**Signature**

dialog-next-page-setter next-page dialog => next-page

**Parameters**

- **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>)`.  

**Discussion**

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**

- `dialog-next-button`
• `dialog-next-button-setter`
• `dialog-next-callback`
• `dialog-next-page`
• `dialog-previous-page-setter`

**dialog-page-changed-callback** Generic function
Returns the page-changed callback of the specified multi-page dialog.

**Signature**
`dialog-page-changed-callback` `dialog` `=>` `callback`

**Parameters**
• `dialog` – An instance of type `<dialog-frame>`.

**Values**
• `callback` – An instance of type `false-or(type-union(<command>,<function>))`.

**Discussion**
Returns 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>`
• `dialog-page-changed-callback-setter`
• `<property-frame>`
• `<wizard-frame>`

**dialog-page-changed-callback-setter** Generic function
Sets the page-changed callback of the specified multi-page dialog.

**Signature**
`dialog-page-changed-callback-setter` `callback` `dialog` `=>` `callback`

**Parameters**
• `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>))`.

**Discussion**
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>`
• `dialog-page-changed-callback`
• `<property-frame>`
• `<wizard-frame>`
**dialog-page-complete? Generic function**

Returns true if all the information required on the current page of the specified multi-page dialog has been specified.

**Signature**  
dialog-page-complete? dialog => complete?

**Parameters**  
- dialog – An instance of type <dialog-frame>.

**Values**  
- complete? – An instance of type <boolean>.

**Discussion**  
Returns true if all the information required on the current page in dialog has been specified by the user. This generic function has two uses:
- 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.
- 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.

**See also**
- **dialog-page-complete?-setter**

**dialog-page-complete?-setter Generic function**

Sets the slot that indicates all the information required on the current page of the specified multi-page dialog has been specified.

**Signature**  
dialog-page-complete? complete? dialog => complete?

**Parameters**  
- complete? – An instance of type <boolean>.
- dialog – An instance of type <dialog-frame>.

**Values**  
- complete? – An instance of type <boolean>.

**Discussion**  
Sets the slot that indicates all the information required on the current page in dialog has been specified by the user. This generic function has two uses:
- 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?**

**dialog-pages Generic function**

Returns the pages of the specified multi-page dialog.

**Signature**  
dialog-pages dialog => pages

**Parameters**  
- dialog – An instance of type <dialog-frame>.
Values

- **pages** – An instance of type `limited(<sequence>, of: <page>).`

Discussion Returns the pages of `dialog`. Each of the items in sequence is an instance of `<page>.

See also

- `<dialog-frame>`
- `dialog-pages-setter`
- `<property-frame>`
- `<wizard-frame>`

**dialog-pages-setter** Generic function

Sets the pages of the specified multi-page dialog.

Signature `dialog-pages-setter pages dialog => pages`

Parameters

- **pages** – An instance of type `limited(<sequence>, of: <page>).`
- **dialog** – An instance of type `<dialog-frame>`.

Values

- **pages** – An instance of type `limited(<sequence>, of: <page>).`

Discussion Sets the pages of `dialog`. Each of the items in sequence must be an instance of `<page>.

See also

- `<dialog-frame>`
- `dialog-pages`
- `<property-frame>`
- `<wizard-frame>`

**dialog-previous-page** Generic function

Returns the previous page in sequence for the specified multi-page dialog.

Signature `dialog-previous-page dialog => previous-page`

Parameters

- **dialog** – An instance of type `<dialog-frame>`.

Values

- **previous-page** – An instance of type `false-or(<page>).`

Discussion Returns the previous 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 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`
dialog-previous-page-setter Generic function
Specifies the previous page in sequence for the specified multi-page dialog.

Signature  dialog-previous-page-setter previous-page dialog => previous-page
Parameters
• previous-page – An instance of type false-or(<page>).
• dialog – An instance of type <dialog-frame>.
Values
• previous-page – An instance of type false-or(<page>).
Discussion
Specifies the previous 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 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
• dialog-back-button-setter
• dialog-back-callback
• dialog-next-page-setter
• dialog-previous-page

display-progress-note Generic function
Displays the specified progress note.

Signature  display-progress-note framem progress-note => ()
Parameters
• framem – An instance of type <frame-manager>.
• progress-note – An instance of type <progress-note>.
Discussion  Displays the specified progress-note in the frame managed by framem.
event-destroy-frame? Generic function
Returns information about the frame was destroyed in the specified event.

Signature  event-destroy-frame? event => destroyed?
Parameters
• event – An instance of type <frame-exit-event>.
Values
• **destroyed?** – An instance of type `<boolean>`.

**Discussion** Returns information about the frame was destroyed in *event*.

**See also**

• `<frame-exit-event>`

**event-status-code** Generic function

Returns the status code of the specified event.

**Signature** `event-status-code event => code`

**Parameters**

• **event** – An instance of type `<frame-exited-event>`.

**Values**

• **code** – An instance of type `false-or(<integer>)`.

**Discussion** Returns the status code of *event*.

**See also**

• `<frame-exited-event>`

**execute-command** Generic function

Executes a command for the specified frame.

**Signature** `execute-command command frame => #rest values`

**Parameters**

• **command** – An instance of type `<command>`.

• **frame** – An instance of type `<frame>`.

**Values**

• **#rest values** – Instances of type `<object>`.

**Discussion** Executes *command* for *frame*. The values returned are those values returned as a result of evaluating the command function of *command*.

**exit-dialog** Generic function

Exits the specified dialog.

**Signature** `exit-dialog dialog #key destroy? => ()`

**Parameters**

• **dialog** – An instance of type `<dialog-frame>`.

**Destroy?** An instance of type `<boolean>`. Default value: `#t`.

**Discussion** Exits *dialog*, recording any changes to the information displayed in the dialog that have been made by the user.

This is the default callback used for the exit button in a dialog. This is the button that is typically labeled *OK*.

If *destroy?* is `#t`, then dialog is destroyed.
Example  The following example defines a button, *yes-button*, that calls \textit{exit-dialog} as its \texttt{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.

\begin{verbatim}
define variable *yes-button* = make(<push-button>, label: "Yes",
activate-callback: exit-dialog,
max-width: $fill);

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*);
\end{verbatim}

See also

\begin{itemize}
\item cancel-dialog
\item <dialog-frame>
\item start-dialog
\end{itemize}

\textbf{exit-frame} Generic function

Unmaps the specified frame destroying it required.

\textbf{Signature} \textsc{exit-frame} frame \#key destroy? \Rightarrow ()

\textbf{Parameters}

\begin{itemize}
\item \texttt{frame} – An instance of type \texttt{<frame>}.
\item \texttt{destroy?} – An instance of type \texttt{<boolean>}. Default value: \#t.
\end{itemize}

\textbf{Discussion}

Unmaps \texttt{frame}, removing the associated sheet and its children from the screen. If \texttt{destroy?} is true, then the frame is destroyed completely, via a call to \texttt{destroy-frame}.

If \texttt{destroy?} is \#t, then dialog is destroyed.

\textbf{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.

\begin{verbatim}
define variable *frame* =
make(<simple-frame>, title: "A frame",
layout: make(<button>)); // RETURN
start-frame(*frame*); // RETURN
exit-frame(*frame*); // RETURN
\end{verbatim}
See also

• destroy-frame
• frame-can-exit?
• <frame-exited-event>
• <frame-exit-event>
• frame-mapped?-setter
• start-frame

find-frame Function
Returns a frame of the specified type, creating one if necessary.

Signature find-frame frame-class #rest initargs #key create? activate? own-thread? port frame-manager test #all-keys => frame

Parameters

• frame-class – An instance of type <object>.
• initargs (#rest) – An instance of type <object>.
• create? (#key) – An instance of type <boolean>. Default value: #t.
• activate? (#key) – An instance of type <boolean>. Default value: #t.
• own-thread? (#key) – An instance of type <boolean>. Default value: #t.
• port (#key) – An instance of type <port>.
• frame-manager (#key) – An instance of type <frame-manager>.
• test (#key) – An instance of type <function>. Default value: identity.

Values

• frame – An instance of type <frame>.

Discussion

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.

The frame-class 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.

The init-args 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.

If create? is #f, then a new frame will not be created if it does not already exist.

If own-thread? is true, the frame will run in its own thread if one is created.

The port and frame-manager 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.

If desired, you can supply a test which must evaluate to true for a frame to match successfully.

See also

• <frame>

<frame> Open Abstract Class
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.
- event-queue – An instance of type false-or(<event-queue>). Default value: #f.
- input-focus – An instance of type false-or(<sheet>). Default value: #f.
- foreground – An instance of type false-or(<ink>).
- background – An instance of type false-or(<int>).
- text-style – An instance of type false-or(<text-style>).
- palette – An instance of type false-or(<palette>). Default value: #f.
- document – An instance of type false-or(<object>). Default value: #f.
- resource-id – An instance of type false-or(<integer>).

Discussion

The class of all frames.

The owner: init-keyword is the parent of the frame.

The mode: 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 `<dialog-frame>`, restricting the user’s interaction with other frames in the application until the modal frame has been dismissed.

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 frames 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 `DUIM-Frames` module.

- `apply-in-frame`
- `call-in-frame`
- `command-enabled?`
- `command-enabled?-setter`
- `deiconify-frame`
- `destroy-frame`
- `execute-command`
- `exit-frame`
- `frame?`
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

Returns true if the specified object is a frame.

**Signature**

\[
\text{frame? } \text{object } \Rightarrow \text{frame?}
\]

**Parameters**

- `object` – An instance of type `<object>`.

**Values**

- `frame?` – An instance of type `<boolean>`.

**Discussion**

Returns true if `object` 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**

- `current-frame`
- `<frame>`

### frame-accelerators Generic function

Returns the keyboard accelerators defined for the specified frame.

**Signature**

\[
\text{frame-accelerators } \text{frame } \Rightarrow \text{accelerators}
\]

**Parameters**

- `frame` – An instance of type `<frame>`.

**Values**

- `accelerators` – An instance of type `false-or(limited(<sequence>, of: <gesture>))`.

**Discussion**

Returns the keyboard accelerators defined for `frame`.

**See also**

- `frame-accelerators-setter`

### frame-accelerators-setter Generic function

Defines the keyboard accelerators for the specified frame.

**Signature**

\[
\text{frame-accelerators } \text{accelerators } \text{frame } \Rightarrow \text{accelerators}
\]

**Parameters**

- `accelerators` – An instance of type `false-or(limited(<sequence>, of: <gesture>))`. 
• **frame** – An instance of type `<frame>`.

**Values**

• **accelerators** – An instance of type `false-or(limited(<sequence>, of:<gesture>))`.

**Discussion** Defines the keyboard accelerators for `frame`.

**See also** `frame-accelerators`

**frame-can-exit?** Open Generic function

Returns true if the specified frame can be exited dynamically.

**Signature** `frame-can-exit? frame => can-exit?`

**Parameters**

• **frame** – An instance of type `<frame>`.

**Values**

• **can-exit?** – An instance of type `<boolean>`.

**Discussion** Returns true if `frame` 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`

**frame-command-table** Generic function

Returns the command table associated with the specified frame.

**Signature** `frame-command-table frame => command-table`

**Parameters**

• **frame** – An instance of type `<frame>`.

**Values**

• **command-table** – An instance of type `<command-table>`.

**Discussion** Returns the command table associated with `frame`.

**See also**

• `frame-command-table-setter`

**frame-command-table-setter** Generic function

Specifies the command table associated with the specified frame.

**Signature** `frame-command-table-setter command-table frame => command-table`

**Parameters**
- **command-table** – An instance of type `<command-table>`.
- **frame** – An instance of type `<frame>`.

Values
- **command-table** – An instance of type `<command-table>`.

Discussion  Specifies the command table associated with `frame`.

See also
- `frame-command-table`

<frame-created-event>  Instantiable Sealed Class
The class of events that indicate a frame has been created.

Superclasses  `<frame-event>`

Discussion  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.

See also
- `<frame-destroyed-event>`
- `<frame-exited-event>`

<frame-destroyed-event>  Instantiable Sealed Class
The class of events that indicate a frame has been destroyed.

Superclasses  `<frame-event>`

Discussion  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.

See also
- `destroy-frame`
- `<frame-created-event>`
- `<frame-exited-event>`

frame-default-button  Generic function
Returns the default button associated with the specified frame.

Signature  `frame-default-button frame => default-button`

Parameters
- `frame` – An instance of type `<frame>`.

Values
- `default-button` – An instance of type `false-or(<button>)`.

Discussion  Returns the default button associated with `frame`.

See also
- `frame-default-button-setter`

frame-default-button-setter  Generic function
Sets the default button associated with the specified frame.

Signature  `frame-default-button-setter default-button frame => default-button`
Parameters

- **default-button** – An instance of type false-or(<button>).
- **frame** – An instance of type <frame>.

Values

- **default-button** – An instance of type false-or(<button>).

Discussion  Sets the default button associated with frame.

See also

- frame-default-button

**frame-event-queue** Generic function

Returns the event queue that the specified frame is running in.

Signature  frame-event-queue frame => event-queue

Parameters

- **frame** – An instance of type <frame>.

Values

- **event-queue** – An instance of type <event-queue>.

Discussion  Returns the event queue that frame is running in.

See also

- <frame>

**<frame-exited-event>** Instantiable Sealed Class

The class of events that indicate a frame has been exited.

Superclasses  <frame-event>

Init-Keywords

- **status-code** – An instance of type false-or(<integer>).

This class also inherits the frame: init-keyword from its superclass.

Discussion

Example  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.

The status-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.

See also

- <application-exited-event>
- exit-frame
- <frame-created-event>
- <frame-destroyed-event>

**<frame-exit-event>** Instantiable Sealed Class

The class of events distributed when a frame is about to exit.

Superclasses  <frame-event>
Init-Keywords

- **destroy-frame?** – An instance of type `<boolean>`. Default value: #f.

Discussion

The class of events distributed when a frame is about to exit. Contrast this with `<frame-exited-event>`, which is passed after the frame is exited.

The default method uses `frame-can-exit?` to decide whether or not to exit.

If `destroy-frame?:` is #t, then the frame is destroyed.

See also

- `event-destroy-frame?`
- `frame-can-exit?`
- `<frame-exited-event>`

**<frame-focus-event>** Instantiable Sealed Class

The class of events distributed when a frame receives focus.

Superclasses `<frame-event>`

Discussion  The class of events distributed when a frame receives the mouse focus.

See also

- `event-destroy-frame?`
- `frame-can-exit?`
- `<frame-exited-event>`

**frame-fixed-height?** Generic function

Returns true if the height of the specified frame is not resizable.

Signature  `frame-fixed-width? frame => fixed-height?`

Parameters

- `frame` – An instance of type `<frame>`.

Values

- **fixed-height?** – An instance of type `<boolean>`.

Discussion  Returns true if the height of `frame` is not resizable.

See also

- `frame-fixed-width?`
- `frame-resizable?`

**frame-fixed-width?** Generic function

Returns true if the width of the specified frame is not resizable.

Signature  `frame-fixed-width? frame => fixed-width?`

Parameters

- `frame` – An instance of type `<frame>`.

Values

- **fixed-width?** – An instance of type `<boolean>`.
**Discussion** Returns true if the width of `frame` is not resizable.

**See also**
- `frame-fixed-height?`
- `frame-resizable?`

**frame-icon** Generic function

Returns the icon associated with the specified frame.

**Signature** `frame-icon frame => icon`

**Parameters**
- `frame` – An instance of type `<frame>`.

**Values**
- `icon` – An instance of type `false-or(<image>)`.

**Discussion** Returns the icon associated with `frame`. 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**
- `deiconify-frame`
- `frame-icon-setter`
- `iconify-frame`

**frame-icon-setter** Generic function

Specifies the icon associated with the specified frame.

**Signature** `frame-icon-setter icon frame => icon`

**Parameters**
- `icon` – An instance of type `false-or(<image>)`.
- `frame` – An instance of type `<frame>`.

**Values**
- `icon` – An instance of type `false-or(<image>)`.

**Discussion** Specifies the icon associated with `frame`. 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**
- `frame-icon`

**frame-input-focus** Generic function

Returns the sheet in the specified frame that has the input focus.

**Signature** `frame-input-focus frame => focus`

**Parameters**
- `frame` – An instance of type `<frame>`.

**Values**
- `focus` – An instance of type `false-or(<sheet>)`. 
Discussion  Returns the sheet in frame that has the input focus.

See also  
  • frame-input-focus-setter

frame-input-focus-setter  Generic function  
Sets which sheet in the specified frame has the input focus.

Signature  frame-input-focus-setter focus frame => focus

Parameters  
  • focus – An instance of type false-or(<sheet>).
  • frame – An instance of type <frame>.

Values  
  • focus – An instance of type false-or(<sheet>).

Discussion  Sets which sheet in frame has the input focus.

See also  
  • frame-input-focus

frame-layout  Generic function  
Returns the layout used in the specified frame.

Signature  frame-layout frame => layout

Parameters  
  • frame – An instance of type <frame>.

Values  
  • layout – An instance of type false-or(<sheet>).

Discussion  Returns the layout used in frame.

See also  
  • frame-layout-setter

frame-layout-setter  Generic function  
Specifies the layout used in the specified frame.

Signature  frame-layout-setter layout frame => layout

Parameters  
  • layout – An instance of type false-or(<sheet>).
  • frame – An instance of type <frame>.

Values  
  • layout – An instance of type false-or(<sheet>).

Discussion  Specifies the layout used in frame.

See also  
  • frame-layout

frame-mapped?  Generic function  
Returns true if the specified frame is mapped.
**Signature**  frame-mapped? frame => mapped?

**Parameters**

- **frame** – An instance of type `<frame>`.

**Values**

- **mapped?** – An instance of type `<boolean>`.

**Discussion** 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 interpreter, pressing the RETURN key at the points indicated.

```scheme
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`

**<frame-mapped-event> Instantiable Sealed Class**

The class of events that indicate a frame has been mapped.

**Superclasses** `<frame-event>`

**Discussion** 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.

**Example** The following example defines a method that can inform you when an instance of a class of frame you have defined is mapped.

```scheme
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>`

**frame-mapped?-setter** Generic function

Maps or unmaps the specified frame.

**Signature**  frame-mapped?-setter mapped? frame => mapped?

**Parameters**
• **mapped?** – An instance of type `<boolean>`.
• **frame** – An instance of type `<frame>`.

Values

• **mapped?** – An instance of type `<boolean>`.

**Discussion** Maps or unmaps `frame`, that is, displays frame on the screen or removes it from the screen, depending on whether `mapped?` 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 `frame-mapped?-setter` rather than `start-frame` and `exit-frame`. 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
frame-mapped?-setter(#t, *frame*); // RETURN
frame-mapped?-setter(#f, *frame*); // RETURN```

See also

• `exit-frame`
• `frame-mapped?`
• `start-frame`

**frame-menu-bar** Generic function

Returns the menu bar used in the specified frame.

**Signature** `frame-menu-bar frame => menu-bar`

**Parameters**

• **frame** – An instance of type `<frame>`.

**Values**

• **menu-bar** – An instance of type `false-or(<menu-bar>)`.

**Discussion** Returns the menu bar used in `frame`.

See also

• `frame-menu-bar-setter`

**frame-menu-bar-setter** Generic function

Sets the menu bar used in the specified frame.

**Signature** `frame-menu-bar-setter menu-bar frame => menu-bar`

**Values**

• **menu-bar** – An instance of type `false-or(<menu-bar>)`.

**Parameters**

• **frame** – An instance of type `<frame>`.

**Discussion** Sets the menu bar used in `frame`. 
See also

- frame-menu-bar

frame-mode Generic function
Returns the mode of the specified frame.

Signature frame-mode frame => mode

Parameters

- **frame** – An instance of type `<frame>`.

Values

- **mode** – An instance of type `one-of(#"modeless", #"modal", #"system-modal")`.

Discussion

Returns the mode of *frame*. This is the same value as was specified for the `mode: init-keyword` when the frame was created.

If *frame* 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).

If *frame* 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.

If *frame* is system-modal, then it prevents the user from interacting with *any* other running applications, such as the Shutdown dialog in Windows 95. System modal frames are rarely used, and should be used with caution.

---

**Note:** You can only set the mode of a frame when it is first created. The mode cannot subsequently be changed.

See also

- `<frame>`

frame-owner Generic function
Returns the controlling frame for the specified frame.

Signature frame-owner frame => owner

Parameters

- **frame** – An instance of type `<frame>`.

Values

- **owner** – An instance of type `false-or(<frame>)`.

Discussion Returns the controlling frame for *frame*. 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 *frame* is not necessarily its direct owner: it may be the owner of *frame*’s owner, and so on, depending on the depth of the hierarchy.

frame-palette Generic function
Returns the palette used in the specified frame.
Signature frame-palette frame => palette

Parameters

• frame – An instance of type <frame>.

Values

• palette – An instance of type <palette>.

Discussion Returns the palette used in frame.

See also

• frame-palette-setter

frame-palette-setter Generic function
Sets the palette used in the specified frame.

Signature frame-palette-setter palette frame => palette

Parameters

• palette – An instance of type <palette>.
  • frame – An instance of type <frame>.

Values

• palette – An instance of type <palette>.

Discussion Sets the palette used in frame.

See also

• frame-palette

frame-position Generic function
Returns the position on the screen of the specified frame.

Signature frame-position frame => x y

Parameters

• frame – An instance of type <frame>.

Values

• x – An instance of type <integer>.
  • y – An instance of type <integer>.

Discussion Returns the position on the screen of frame. 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.

```lisp
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
• *frame-state*

*set-frame-position*

**frame-resizable?** Generic function

Returns true if the specified frame is resizable.

**Signature**  
frame-resizable? frame => resizable?

**Parameters**

- *frame* – An instance of type <frame>.

**Values**

- *resizable?* – An instance of type <boolean>.

**Discussion** Returns true if *frame* is resizable, that is can have one or both of its width and height modified by the user.

**See also**

- *frame-fixed-height?*
- *frame-fixed-width?*

**frame-size** Generic function

Returns the size of the specified frame.

**Signature**  
frame-size frame => width height

**Parameters**

- *frame* – An instance of type <frame>.

**Values**

- *width* – An instance of type <integer>.
- *height* – An instance of type <integer>.

**Discussion** Returns the size of *frame*, 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.

```lisp
define variable *frame* =  
make(<simple-frame>, title: "A frame",  
layout: make(<button>)); // RETURN  
start-frame(*frame*); // RETURN  
frame-size(*frame*); // RETURN
```

**See also**

- *frame-position*
- *frame-state*
- *set-frame-size*

**frame-state** Generic function

Returns the visible state of the specified frame.

**Signature**  
frame-state frame => state

**Parameters**

- *frame* – An instance of type <frame>. 

10.4. DUIM-Frames Module
Values

• **state** – An instance of type one-of(#"detached", #"unmapped", #"mapped", #"iconified", #"destroyed").

**Discussion** 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.

```lisp
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**

• **frame-size**

**frame-status-bar** Generic function

Returns the status bar used in the specified frame.

**Signature** frame-status-bar frame => status-bar

**Parameters**

• **frame** – An instance of type <frame>.

**Values**

• **status-bar** – An instance of type false-or(<status-bar>).

**Discussion** Returns the status bar used in frame.

**See also**

• **frame-status-bar-setter**

**frame-status-bar-setter** Generic function

Sets the status bar used in the specified frame.

**Signature** frame-status-bar-setter status-bar frame => status-bar

**Parameters**

• **status-bar** – An instance of type <status-bar>.

• **frame** – An instance of type <frame>.

**Values**

• **status-bar** – An instance of type false-or(<status-bar>).

**Discussion** Sets the status bar used in frame.

**See also**

• **frame-status-bar**

**frame-status-message** Open Generic function

Returns the status message for the specified frame.
Signature  frame-status-message frame => status-message

Parameters
  • frame – An instance of type <frame>.

Values
  • status-message – An instance of type false-or(<string>).

Discussion  Returns the status message for frame. 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
  • frame-status-bar
  • frame-status-message-setter
  • <status-bar>

frame-status-message-setter  Generic function
  Sets the status message for the specified frame.

Signature  frame-status-message status-message frame => status-message

Parameters
  • status-message – An instance of type false-or(<string>).
  • frame – An instance of type <frame>.

Values
  • status-message – An instance of type false-or(<string>).

Discussion  Sets the status message for frame. 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
  • frame-status-bar-setter
  • frame-status-message
  • <status-bar>

frame-thread  Generic function
  Returns the thread with which the specified frame is associated.

Signature  frame-thread frame => thread

Parameters
  • frame – An instance of type <frame>.

Values
  • thread – An instance of type <thread>.

Discussion  Returns the thread with which frame is associated.

For more information about threads, refer to the manual Library Reference: Core Features.

frame-title  Generic function
  Returns the title of the specified frame.

Signature  frame-title frame => title
Parameters

- **frame** – An instance of type `<frame>`.

Values

- **title** – An instance of type `false-or(<string>)`.

Discussion Returns the title of `frame`. 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

- **frame-title-setter**

**frame-title-setter** Generic function

Sets the title of the specified frame.

Signature  
```
frame-title-setter title frame => title
```

Parameters

- **title** – An instance of type `false-or(<string>)`.
- **frame** – An instance of type `<frame>`.

Values

- **title** – An instance of type `false-or(<string>)`.

Discussion Sets the title of `frame`. The title of a frame is displayed in the title bar of the frame. If `title` is `#f`, then the platform attempts to remove the title bar from the frame, if possible.

See also

- **frame-title**

**frame-tool-bar** Generic function

Returns the tool bar used in the specified frame.

Signature  
```
frame-tool-bar frame => tool-bar
```

Parameters

- **frame** – An instance of type `<frame>`.

Values

- **tool-bar** – An instance of type `false-or(<tool-bar>)`.

Discussion Returns the tool bar used in `frame`.

See also

- **frame-tool-bar-setter**

**frame-tool-bar-setter** Generic function

Sets the tool bar used in the specified frame.

Signature  
```
frame-tool-bar-setter tool-bar frame => tool-bar
```

Parameters

- **tool-bar** – An instance of type `false-or(<tool-bar>)`.
- **frame** – An instance of type `<frame>`.

Values
• **tool-bar** – An instance of type \texttt{false-or(<tool-bar>)}.

**Discussion** Sets the tool bar used in \textit{frame}.

**See also**

• \texttt{frame-tool-bar}

**frame-top-level** Generic function

Returns the top level loop function for the specified frame.

**Signature** \texttt{frame-top-level frame => top-level}

**Parameters**

• \texttt{frame} – An instance of type \texttt{<frame>}.

**Values**

• \texttt{top-level} – An instance of type \texttt{<function>}.

**Discussion**

Returns the top level loop function for \textit{frame}. The top level loop function for a frame is the “command loop” for the frame.

The default method for \texttt{frame-top-level} calls \texttt{read-event} and then \texttt{handle-event}.

**See also**

• \texttt{handle-event}

**<frame-unmapped-event>** Instantiable Sealed Class

The class of events that indicate a frame has been unmapped.

**Superclasses** \texttt{<frame-event>}

**Discussion** 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.

**Example** The following example defines a method that can inform you when an instance of a class of frame you have defined is unmapped.

```lisp
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**

• \texttt{<frame-mapped-event>}

**global-command-table** Variable

The command table inherited by all new command tables.

**Type** \texttt{<command-table>}

**Discussion** 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

- `<command-table>`
- `*user-command-table*`

**iconify-frame** Generic function

Iconifies the specified frame.

**Signature**

`iconify-frame frame => ()`

**Parameters**

- `frame` – An instance of type `<frame>`.

**Discussion**

Iconifies `frame`. 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.

```lisp
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`
- `destroy-frame`
- `exit-frame`
- `frame-icon`
- `lower-frame`
- `raise-frame`

**layout-frame** Generic function

Resizes the specified frame and lays out the current pane hierarchy inside it.

**Signature**

`layout-frame frame #key width height => ()`

**Parameters**

- `frame` – An instance of type `<frame>`.
- `width` – An instance of type `false-or(<integer>)`.
- `height` – An instance of type `false-or(<integer>)`.

**Discussion**

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
Lowers the specified frame to the bottom of the stack of visible windows.

**Signature**
```plaintext```
lower-frame frame => ()
```plaintext```

**Parameters**
- `frame` – An instance of type `<frame>`.

**Discussion**
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.

```plaintext```
define variable *frame* =
  make(<simple-frame>, title: "A frame",
       layout: make(<button>));
start-frame(*frame*);
lower-frame(*frame*);
```plaintext```

**See also**
- `deiconify-frame`
- `destroy-frame`
- `exit-frame`
- `iconify-frame`
- `raise-frame`

**make (<frame>)** Method
Creates an instance of a `<frame>`.

**Signature**
```plaintext```
make (class == <frame>) #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 => simple-frame
```plaintext```

**Parameters**
- `class` – The class `<frame>`.
- `top-level (#key)` – An instance of type `false-or (<sheet>)`. Default value: `#f`.
- `command-queue (#key)` – An instance of type `false-or (<event-queue>)`. Default value: `#f`.
- `layout (#key)` – An instance of type `false-or (<sheet>)`. Default value: `#f`.
- `icon (#key)` – An instance of type `false-or (<image>)`. Default value: `#f`.
- `pointer-documentation (#key)` – An instance of type `false-or (<string>)`. Default value: `#f`.
- `command-table (#key)` – An instance of type `false-or (<command-table>)`. Default value: `#f`.
- `menu-bar (#key)` – An instance of type `false-or (<menu-bar>)`. Default value: `#f`.

10.4. DUIM-Frames Module
• **tool-bar** (#key) – An instance of type `false-or(tool-bar)`. Default value: #f.

• **status-bar** (#key) – An instance of type `false-or(status-bar)`. Default value: #f.

• **title** (#key) – An instance of type `false-or(string)`. Default value: #f.

• **calling-frame** (#key) – An instance of type `false-or(frame)`. Default value: #f.

• **state** (#key) – An instance of type `one-of(detached, unmapped, mapped, iconified)`. Default value: "detached".

• **geometry** (#key) – An instance of type `vector()`. Default value: `vector(#f, #f, #f, #f)`.

• **resizable?** (#key) – An instance of type `boolean`. Default value: #t.

• **properties** (#key) – An instance of type `stretchy-object-vector`. Default value: `make(stretchy-vector)`.

• **thread** (#key) – An instance of type `false-or(thread)`. Default value: #f.

• **event-queue** (#key) – An instance of type `false-or(event-queue)`. Default value: #f.

• **foreground** (#key) – An instance of type `false-or(ink)`. Default value: #f.

• **background** (#key) – An instance of type `false-or(ink)`. Default value: #f.

• **text-style** (#key) – An instance of type `false-or(text-style)`. Default value: #f.

• **palette** (#key) – An instance of type `false-or(palette)`. Default value: #f.

• **save-under?** (#key) – An instance of type `boolean`. Default value: #f.

• **drop-shadow?** (#key) – An instance of type `boolean`. Default value: #f.

• **dialog-for** (#key) – An instance of type `dialog-frame`.

**Values**

• **simple-frame** – An instance of type `<frame>`.

**Discussion**

Creates and returns an instance of `<frame>` or one of its subclasses.

The **top-level** argument specifies the top-level command-loop in which the frame runs.

The **command-queue** argument specifies a command-queue for the frame.

The **layout** argument specifies a layout for the frame.

The **icon** 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.

The **pointer-documentation** argument specifies pointer-documentation for the frame.

The **command-table** argument specifies a command table for the frame.

The **menu-bar** argument specifies a menu bar for the frame.

The **tool-bar** argument specifies a tool bar for the frame.
The status-bar argument specifies a status bar for the frame.
The title argument specifies a title for the frame.
The calling-frame argument specifies a calling frame for the frame.
The state argument specifies a frame-state. The frame can be mapped or unmapped (that is, visible on the screen, or not), iconified, or detached.
The geometry 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.
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>

make-menu-from-command-table-menu Generic function
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

Parameters
• command-table-menu-items – An instance of type <sequence>.
• frame – An instance of type <frame>.
• framem – An instance of type <frame-manager>.
• command-table (#key) – An instance of type <command-table>.
• label (#key) – An instance of type <label>.
• mnemonic (#key) – An instance of type false-or(<gesture>).
• item-callback (#key) – An instance of type <function>.

Values
• menu – An instance of type <menu>.

Discussion
Returns a menu from the menu definition in the specified command table. This function is used by make-menus-from-command-table to individually create each menu defined in the command table. The function make-menus-from-command-table then puts each of the menus created together in the appropriate way.

The command-table-menu-items argument defines the items that are to be placed in the menu. It is a sequence of instances of <command-table-menu-item>.

The frame and framem arguments define the frame and the frame manager in which the menu created is to be placed.
The command-table argument specifies the command table in which the definition of the menu created can be found.

The label argument defines a label for the menu created.

The mnemonic argument defines a keyboard mnemonic for the menu created.

See also

- make-menus-from-command-table

make-menus-from-command-table Generic function
Returns a set of menus from the menu definitions in the specified command table.

Signature make-menus-from-command-table command-table frame framem #key label => menus

Parameters

- command-table – An instance of type <command-table>.
- frame – An instance of type <frame>.
- framem – An instance of type <frame-manager>.
- label(#key) – An instance of type <label>.

Values

- menus – An instance of type limited(<sequence>, of: <menu>).

Discussion

Returns a set of menus from the menu definitions in command-table.

The frame and framem arguments specify the frame and frame manager in which the menus are to be placed.

The label argument lets you specify a label for the set of menus.

See also make-menu-from-command-table

menu-item-accelerator Generic function
Returns the accelerator for the specified command table menu item.

Signature menu-item-accelerator menu-item => accelerator

Parameters

- menu-item – An instance of type <command-table-menu-item>.

Values

- accelerator – An instance of type <gesture>.

Discussion Returns the keyboard accelerator for menu-item. Note that menu-item must be defined in a command table.

See also

- menu-item-mnemonic

menu-item-mnemonic Generic function
Returns the mnemonic for the specified menu item.

Signature menu-item-mnemonic menu-item => mnemonic

Parameters

- menu-item – An instance of type <command-table-menu-item>.
Values

- **mnemonic** – An instance of type `false-or(<gesture>)`.

Discussion Returns the keyboard mnemonic for `menu-item`.

See also

- `menu-item-accelerator`

### `menu-item-name` Generic function

Returns the name of the specified menu item.

**Signature**

```
menu-item-name menu-item => name
```

**Parameters**

- **menu-item** – An instance of type `<command-table-menu-item>`.

**Values**

- **name** – An instance of type `<string>`.

Discussion Returns the name of `menu-item`.

See also

- `menu-item-options`
- `menu-item-type`
- `menu-item-value`

### `menu-item-options` Generic function

Returns the options for the specified menu item.

**Signature**

```
menu-item-options menu-item => options
```

**Parameters**

- **menu-item** – An instance of type `<command-table-menu-item>`.

**Values**

- **options** – An instance of type `<object>`.

Discussion Returns the options for `menu-item`.

See also

- `menu-item-name`
- `menu-item-type`
- `menu-item-value`

### `menu-item-type` Generic function

Returns the type of the specified menu item.

**Signature**

```
menu-item-type menu-item => type
```

**Parameters**

- **menu-item** – An instance of type `<command-table-menu-item>`.

**Values**

- **type** – An instance of type `<object>`.

Discussion Returns the type of `menu-item`.
menu-item-value Generic function

Returns the value of the specified menu item.

Signature  menu-item-value menu-item => value

Parameters

- • menu-item – An instance of type <command-table-menu-item>.

Values

- • value – An instance of type <object>.

Discussion  Returns the value of menu-item.

See also

- • menu-item-name
- • menu-item-options
- • menu-item-type

move-to-next-page Generic function

Moves to the next page of the specified multi-page dialog.

Signature  move-to-next-page wizard => ()

Parameters

- • wizard – An instance of type <wizard-frame>.

Discussion  Moves to the next page in sequence of wizard. This is the default callback for the Next button in a wizard frame.

See also

- • dialog-next-callback
- • <wizard-frame>

move-to-previous-page Generic function

Moves to the previous page of the specified multi-page dialog.

Signature  move-to-previous-page wizard => ()

Parameters

- • wizard – An instance of type <wizard-frame>.

Discussion  Moves to the previous page in sequence of wizard. This is the default callback for the Back button in a wizard frame.

See also

- • dialog-back-callback
- • <wizard-frame>

note-progress Generic function

Note the progress of an event in the specified progress note.
Signature  note-progress numerator denominator #key note label pointer-cursor => ()

Parameters

- **numerator** – An instance of type `<integer>`.
- **denominator** – An instance of type `<integer>`.
- **note (#key)** – An instance of type `<progress-note>`. Default value: *progress-note*.
- **label (#key)** – An instance of type `<label>`.
- **pointer-cursor (#key)** – An instance of type `<pointer>`.

Discussion

Note the progress of an event in `note`.

If a `numerator` and `denominator` are supplied, then the progress is displayed in terms of those figures. For example, if `numerator` is 1, and `denominator` is 10, then the progress is displayed in tenths.

If supplied, `pointer-cursor` is used as a cursor when the mouse pointer is placed over the owner frame.

See also

- **noting-progress**
- ***progress-note*”

**noting-progress** Statement Macro

Performs a body of code, noting its progress.

**Macro Call**

```
noting-progress {{*sheet* }, {{*label* }} }{*body* } end
```

Parameters

- **sheet** – A Dylan expression `bnf`.
- **label** – A Dylan expression `bnf`.
- **body** – A Dylan body `bnf`.

Discussion

Performs a body of code, noting its progress, for the specified sheet.

The sheet argument is an expression that evaluates to an instance of `<sheet>`. The label argument is an expression that evaluates to an instance of `<string>`.

See also

- **note-progress**

**progress-note** Thread Variable

Specifies a default progress note that can be used.

**Type** `<object>`

**Value** `#f`

Discussion This variable is used to supply a default progress note to use if no progress note is explicitly specified.
See also

• note-progress

<property-frame> Open Instantiable Class
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>:

Init-Keywords

• pages – An instance of type false-or(<sequence>). Default value: #f.

• page-changed-callback – An instance of type false-or(<function>). Default value: #f.

Discussion

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.

Fig. 10.2: A property frame

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.

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

• 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`
• `dialog-apply-callback`
• `<dialog-frame>`
• `<property-page>`
• `<wizard-frame>`

<property-page> Open Instantiable Class
The class of property pages.

**Superclasses** `<page>`

**Discussion**

The class of property pages. These are pages that can be displayed in an instance of `<property-frame>`.

![Fig. 10.3: A property page](image)

Internally, this class maps into the Windows property page control.

See also

• `<page>`
• `<property-frame>`
• `<property-page>`
• `<tab-control-page>`
• `<wizard-page>`

**raise-frame** Generic function

Raises the specified frame to the top of the stack of visible windows.

**Signature** `raise-frame frame => ()`

**Parameters**
• **frame** – An instance of type `<frame>`.

**Discussion**  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.

```clojure
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`
• `destroy-frame`
• `exit-frame`
• `iconify-frame`
• `lower-frame`

**redo-command**  Generic function

Performs the last performed command again.

**Signature**  redo-command `command frame => #rest values`

**Parameters**

• **command** – An instance of type `<command>`.
• **frame** – An instance of type `<frame>`.
• **values** – Instances of type `<object>`.

**Discussion**  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`

**remove-command**  Generic function

Removes a command from the specified command table.

**Signature**  remove-command `command-table command => ()`

**Parameters**

• **command-table** – An instance of type `<command-table>`.
• **command** – An instance of type `<command>`.

**Discussion**  Removes `command` from `command-table`.

See also
• *add-command*

**remove-command-table** **Function**
Removes the specified command table.

**Signature** `remove-command-table command-table => ()`

**Parameters**
• *command-table* – An instance of type `<command-table>`.

**Discussion** Removes `command-table`.

**remove-command-table-menu-item** **Generic function**
Removes a menu item from the specified command table.

**Signature** `remove-command-table-menu-item command-table string => ()`

**Parameters**
• *command-table* – An instance of type `<command-table>`.
• *string* – An instance of type `<string>`.

**Discussion** Removes the menu item identified by `string` from `command-table`.

See also
• *add-command-table-menu-item*

**set-frame-position** **Generic function**
Sets the position of the specified frame.

**Signature** `set-frame-position frame x y => ()`

**Parameters**
• *frame* – An instance of type `<frame>`.
• *x* – An instance of type `<integer>`.
• *y* – An instance of type `<integer>`.

**Discussion** Sets the position of `frame`. The coordinates `x` and `y` are measured from the top left of the screen, measured in pixels.

See also
• *frame-position*
• *set-frame-size*

**set-frame-size** **Generic function**
Sets the size of the specified frame.

**Signature** `set-frame-size frame width height => ()`

**Parameters**
• *frame* – An instance of type `<frame>`.
• *width* – An instance of type `<integer>`.
• *height* – An instance of type `<integer>`.

**Discussion** 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.
See also

• `frame-size`
• `set-frame-position`

<simple-command> Open Abstract Instantiable Class
The class of simple commands.

Superclasses `<object`

Init-Keywords

• `function` – An instance of type `<function>`. Required.
• `arguments` – An instance of type `<sequence>`. Default value `[]`.

Discussion

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.

See also

• `<command>`

<simple-frame> Open Abstract Instantiable Class
The class of simple frames.

Superclasses `<frame`

Init-Keywords

• `command-queue` – An instance of type `false-or(<event-queue>)`. Default value: `make(<event-queue>)`.
• `layout` – An instance of type `false-or(<sheet>)`. Default value: `""#f`.
• `command-table` – An instance of type `false-or(<command-table>)`. Default value: `#f`.
• `menu-bar` – An instance of type `false-or(<menu-bar>)`. Default value: `#f`.
• `tool-bar` – An instance of type `false-or(<tool-bar>)`. Default value: `#f`.
• `status-bar` – An instance of type `false-or(<status-bar>)`. Default value: `#f`.

Discussion

The class of simple frames.

The command-queue: 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

• `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
The class of simple commands that can contain an undo action.

Superclasses <object>

Init-Keywords
• undo-command – An instance of type <command>.

Discussion 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.

See also
• <simple-command>

start-dialog Generic function
Displays a DUIM frame as a dialog box.

Signature start-dialog dialog => #rest values

Parameters
• dialog – An instance of type <dialog-frame>.

Values
• #rest values – Instances of type <object>.

Discussion Displays a DUIM frame as a dialog box.

The function start-dialog dynamically binds an <abort> 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 <abort> (via the abort 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.

See also
DUIM Reference Documentation, Release 1.0

• cancel-dialog
• <dialog-frame>
• exit-dialog
• start-frame

start-frame Generic function
Starts the specified frame.

Signature start-frame frame #key owner mode => status-code

Parameters

• frame – An instance of type <frame>.
• owner – An instance of type false-or(<frame>). Default value: #f.
• mode – An instance of type one-of("modal", #"modeless", #"system-modal"). Default value: #f.

Values
• status-code – An instance of type <integer>.

Discussion
Starts frame, optionally setting the owner of the frame and the mode in which it will run.
The function start-frame dynamically binds an <abort> 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 <abort> (via the abort 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.

Example 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.

```
define variable *frame* =
  make(<simple-frame>, title: "A frame",
       layout: make(<button>)); // RETURN

start-frame(*frame*); // RETURN
```

See also
• exit-frame
• frame-mapped?-setter
• start-dialog

undo-command Generic function
Calls the undo command for the specified command.

Signature undo-command command frame => #rest values

Parameters

• command – An instance of type <command>.
• frame – An instance of type <frame>.

Values
• **#rest values** – Instances of type `<object>`.

Discussion

Calls the undo command for `command`, undoing the effects of calling `command`. Note that `command` must be undoable.

You can call this command directly in your own code, as well as specialize it.

See also

• `command-undoable?`

**user-command-table** Variable

A user-defined command table that can be inherited by other command tables.

Type `<command-table>`

Discussion

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

• `<command-table>`

• `*global-command-table*`

**<wizard-frame>** Class

Open

Instantiable

The class of wizard frames.

Superclasses `<dialog-frame>`

Init-Keywords

• `page` – An instance of type `<page>`.

• `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 that the following two useful init-keywords are inherited from `<dialog-frame>`:

Init-Keywords

• `pages` – An instance of type `false-or(<sequence>)`. Default value: #f.

• `page-changed-callback` – An instance of type `false-or(<function>)`. Default value: #f.

Discussion

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.
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.

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**

- `compute-next-page`
- `compute-previous-page`
- `dialog-back-button`
- `dialog-back-button-setter`
- `dialog-back-callback`
- `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`
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>`
- `<wizard-page>`

<wizard-page> Open Instantiable Class
The class of wizard pages.

Superclasses `<page>`

---

10.4. DUIM-Frames Module
Discussion

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.

Fig. 10.5: A wizard page

See also

- `<page>`
- `<property-page>`
- `<tab-control-page>`
- `<wizard-frame>`
INDICES AND TABLES

- genindex
- search
duim-dcs: (generic function), 70
duim-geometry: (region) (method), 10
duim-geometry: (transform) (method), 10
duim-sheets: (generic function), 120
duim-frames: (command) (method), 390

$alt-key (constant), 121
$action-gadget (class), 291
$application-exited-event (class), 393
$area (class), 11
abort-path (generic function), 226
activate-gadget (generic function), 292
add-child (generic function), 120
add-clipboard-data (generic function), 121
add-clipboard-data-as (generic function), 121
add-colors (generic function), 71
add-column (generic function), 292
add-command (generic function), 390
add-command-table-menu-item (generic function), 391
add-item (generic function), 292
add-node (generic function), 293
allocate-space (generic function), 255
apply-in-frame (generic function), 393
arc-to (generic function), 226
area? (generic function), 11

$background (constant), 71
$black (constant), 71
$blue (constant), 71
$boole-1 (constant), 72
$boole-2 (constant), 72
$boole-and (constant), 72
$boole-andc1 (constant), 73
$boole-andc2 (constant), 73
$boole-cl (constant), 72
$boole-c2 (constant), 72
$boole-cir (constant), 71
$boole-eqv (constant), 72
$boole-ior (constant), 72
$boole-nand (constant), 73
$boole-nor (constant), 73
$boole-orc1 (constant), 73
$boole-orc2 (constant), 73
$boole-set (constant), 71
$boole-xor (constant), 72
$bricks-stipple (constant), 73
$basic-user-pane (class), 255
$border (class), 293
$bounding-box (class), 255
$brush (class), 73
$button-box (class), 296
$button-press-event (class), 123
$button-release-event (class), 123
$button (class), 294
beep (generic function), 122
boundary-event-kind (generic function), 122
bounding-box (generic function), 12
bounding-box? (generic function), 12
box-bottom (function), 13
box-edges (generic function), 13
box-height (function), 14
box-left (function), 14
box-position (generic function), 15
box-right (function), 15
box-size (generic function), 15
box-top (function), 16
box-width (function), 16
brush-background (generic function), 75
brush-fill-rule (generic function), 75
brush-fill-style (generic function), 75
brush-foreground (generic function), 75
brush-mode (generic function), 76
brush-stipple (generic function), 76
brush-stretch-mode (generic function), 76
brush-tile (generic function), 76
brush-ts-x (generic function), 77
brush-ts-y (generic function), 77
brush? (generic function), 74
button-index (function), 122
button-index-name (function), 122

color (generic function), 79
command-arguments (generic function), 396
command-enabled? (generic function), 396
command-enabled?-setter (generic function), 397
command-function (generic function), 397
command-table-accelerators (generic function), 399
command-table-commands (generic function), 399
command-table-menu (generic function), 400
command-table-name (generic function), 401
command-table? (generic function), 399
command-undoable? (generic function), 401
command? (generic function), 396
complete-from-generator (generic function), 402
complete-from-sequence (generic function), 403
compose-rotation-with-transform (generic function), 16
compose-scaling-with-transform (generic function), 17
compose-space (generic function), 258
compose-transform-with-rotation (generic function), 18
compose-transform-with-scaling (generic function), 18
compose-transform-with-translation (generic function), 19
compose-transforms (generic function), 17
compose-translation-with-transform (generic function), 19
compute-next-page (generic function), 404
compute-previous-page (generic function), 404
contain (generic function), 405
contract-node (generic function), 302
contrasting-colors-limit (generic function), 81
contrasting-dash-patterns-limit (generic function), 81
copy-area (generic function), 229
copy-from-pixmap (generic function), 230
copy-to-pixmap (generic function), 230
current-frame (function), 406
current-pane (generic function), 258
cursor? (generic function), 137
curve-to (generic function), 226

d$
$dash-dot-dot-pen (constant), 82
$dash-dot-pen (constant), 82
$dashed-pen (constant), 83
$diagonal-hatch-down (constant), 85
$diagonal-hatch-up (constant), 85
$dotted-pen (constant), 86
<device-event> (class), 138
<device-font> (class), 85
<dialog-frame> (class), 420
<display> (class), 138
<double-click-event> (class), 146
<drawing-pane> (class), 261
default-background (generic function), 83
default-background-setter (generic function), 83
default-foreground (generic function), 84
default-foreground-setter (generic function), 84
default-port (function), 137
default-port-setter (function), 137
default-text-style (generic function), 84
default-text-style-setter (generic function), 85
define command-table (macro), 406
define frame (macro), 408
define pane (macro), 259
deiconify-frame (generic function), 410
destroy-frame (generic function), 411
destroy-pixmap (generic function), 232
destroy-port (generic function), 138
destroy-sheet (generic function), 138
dialog-apply-button (generic function), 411
dialog-apply-button-setter (generic function), 412
dialog-apply-callback (generic function), 412
dialog-back-button (generic function), 413
dialog-back-button-setter (generic function), 413
dialog-back-callback (generic function), 413
dialog-cancel-button (generic function), 414
dialog-cancel-button-setter (generic function), 414
dialog-cancel-callback (generic function), 415
dialog-cancel-callback-setter (generic function), 415
dialog-current-page (generic function), 416
dialog-current-page-setter (generic function), 416
dialog-exit-button (generic function), 416
dialog-exit-button-setter (generic function), 417
dialog-exit-callback (generic function), 418
dialog-exit-callback-setter (generic function), 418
dialog-exit-enabled? (generic function), 419
dialog-exit-enabled?-setter (generic function), 419
dialog-help-button (generic function), 422
dialog-help-button-setter (generic function), 423
dialog-help-callback (generic function), 424
dialog-next-button (generic function), 424
dialog-next-button-setter (generic function), 425
dialog-next-callback (generic function), 425
dialog-next-enabled? (generic function), 426
dialog-next-enabled?-setter (generic function), 426
dialog-next-page (generic function), 427
dialog-next-page-setter (generic function), 427
dialog-page-changed-callback (generic function), 428
dialog-page-changed-callback-setter (generic function), 428
dialog-page-complete? (generic function), 428
dialog-page-complete?-setter (generic function), 429
dialog-pages (generic function), 429
dialog-pages-setter (generic function), 430
dialog-previous-page (generic function), 430
dialog-previous-page-setter (generic function), 431
display (generic function), 139
display-depth (generic function), 140
display-height (generic function), 140
display-menu (generic function), 302
display-mm-height (generic function), 141
display-mm-width (generic function), 141
display-orientation (generic function), 142
display-pixel-height (generic function), 142
display-pixel-width (generic function), 143
display-pixels-per-point (generic function), 142
display-progress-note (generic function), 431
display-units (generic function), 143
display-width (generic function), 143
display? (generic function), 140
do-allocate-space (generic function), 259
do-children-containing-position (generic function), 144
do-children-overlapping-region (generic function), 144
do-compose-space (generic function), 260
do-coordinates (function), 20
do-displays (function), 145
do-endpoint-coordinates (function), 20
do-frames (generic function), 145
do-polygon-coordinates (generic function), 48
do-polygon-segments (generic function), 48
do-ports (function), 145
do-regions (generic function), 20
do-sheet-children (generic function), 145
do-sheet-tree (generic function), 145
do-with-drawing-options (generic function), 146
do-with-output-to-pixmap (generic function), 232
do-with-pointer-grabbed (generic function), 146
do-with-sheet-medium (generic function), 147
do-with-text-style (generic function), 147
draw-arrow (generic function), 232
draw-bezier-curve (generic function), 234
draw-circle (generic function), 234
draw-design (generic function), 48
draw-ellipse (generic function), 235
draw-image (generic function), 236
draw-line (generic function), 237
draw-lines (generic function), 238
draw-oval (generic function), 238
draw-pixmap (generic function), 239
draw-point (generic function), 240
draw-points (generic function), 241
draw-polygon (generic function), 241
draw-rectangle (generic function), 242
draw-rectangles (generic function), 243
draw-regular-polygon (generic function), 244
draw-text (generic function), 245
draw-triangle (generic function), 246
event? (generic function), 148
execute-command (generic function), 432
exit-dialog (generic function), 432
exit-frame (generic function), 433
expand-node (generic function), 302

F
$fill (constant), 261
$foreground (constant), 90
<fixed-layout> (class), 261
<frame-created-event> (class), 440
<frame-destroyed-event> (class), 440
<frame-event> (class), 156
<frame-exit-event> (class), 441
<frame-exited-event> (class), 441
<frame-focus-event> (class), 442
<frame-manager> (class), 156
<frame-mapped-event> (class), 445
<frame-unmapped-event> (class), 453
<frame> (class), 434
fill-path (generic function), 226
find-color (generic function), 86
find-display (function), 152
find-frame (function), 434
find-frame-manager (function), 152
find-item (generic function), 303
find-node (generic function), 303
find-port (function), 153
fix-coordinate (function), 21
fixed-width-font? (generic function), 153
font-ascent (generic function), 153
font-descent (generic function), 154
font-height (generic function), 154
font-metrics (generic function), 155
font-width (generic function), 155
force-display (generic function), 156
frame-accelerators (generic function), 438
frame-accelerators-setter (generic function), 438
frame-can-exit? (generic function), 439
frame-command-table (generic function), 439
frame-command-table-setter (generic function), 439
frame-default-button (generic function), 440
frame-default-button-setter (generic function), 440
frame-event-queue (generic function), 440
frame-fixed-height? (generic function), 442
frame-fixed-width? (generic function), 442
frame-icon (generic function), 443
frame-icon-setter (generic function), 443
frame-input-focus (generic function), 443
frame-input-focus-setter (generic function), 444
frame-layout (generic function), 444
frame-layout-setter (generic function), 444
frame-manager (generic function), 157
frame-manager-frames (generic function), 158
frame-manager-palette (generic function), 158
frame-manager-palette-setter (generic function), 158
frame-manager? (generic function), 157
frame-mapped? (generic function), 444
frame-mapped?-setter (generic function), 445
frame-menu-bar (generic function), 446
frame-menu-bar-setter (generic function), 446
frame-mode (generic function), 447
frame-owner (generic function), 447
frame-palette (generic function), 447
frame-palette-setter (generic function), 448
frame-position (generic function), 448
frame-resizable? (generic function), 449
frame-size (generic function), 449
frame-state (generic function), 449
frame-status-bar (generic function), 450
frame-status-bar-setter (generic function), 450
frame-status-message (generic function), 450
frame-status-message-setter (generic function), 451
frame-thread (generic function), 451
frame-title (generic function), 451
frame-title-setter (generic function), 452
frame-tool-bar (generic function), 452
frame-tool-bar-setter (generic function), 452
frame-top-level (generic function), 453
frame? (generic function), 453
fully-merged-text-style? (generic function), 90

gadget-default? (generic function), 309
gadget-default?-setter (generic function), 310
gadget-documentation (generic function), 310
gadget-documentation-setter (generic function), 311
gadget-enabled? (generic function), 311
gadget-enabled?-setter (generic function), 312
gadget-id (generic function), 312
gadget-id-setter (generic function), 313
gadget-items (generic function), 314
gadget-items-setter (generic function), 314
gadget-key-press-callback (generic function), 315
gadget-key-press-callback-setter (generic function), 315
gadget-label (generic function), 316
gadget-label-key (generic function), 316
gadget-label-setter (generic function), 317
gadget-mnemonic (generic function), 317
gadget-mnemonic-setter (generic function), 318
gadget-orientation (generic function), 318
gadget-popup-menu-callback (generic function), 318
gadget-popup-menu-callback-setter (generic function), 319
gadget-ratios (generic function), 319
gadget-ratios-setter (generic function), 319
gadget-read-only? (generic function), 320
gadget-scrolling-horizontal? (generic function), 320
gadget-scrolling-vertically? (generic function), 320
gadget-selection (generic function), 320
gadget-selection-mode (generic function), 321
gadget-selection-setter (generic function), 322
gadget-slug-size (generic function), 322
gadget-slug-size-setter (generic function), 323
gadget-test (generic function), 323
gadget-text (generic function), 324
gadget-text-setter (generic function), 324
gadget-value (generic function), 325
gadget-value-changed-callback (generic function), 326
gadget-value-changed-callback-setter (generic function), 326
gadget-value-changing-callback (generic function), 326
gadget-value-changing-callback-setter (generic function), 327
gadget-value-key (generic function), 327
gadget-value-range (generic function), 328

G
$green (constant), 90
+global-command-table+ (variable), 453
<gadget> (class), 304
<gesture> (class), 158
<grid-layout> (class), 262
<group-box> (class), 331
gadget-accelerator (generic function), 306
gadget-accelerator-setter (generic function), 306
gadget-activate-callback (generic function), 307
gadget-activate-callback-setter (generic function), 307
gadget-client (generic function), 308
gadget-client-setter (generic function), 308
gadget-command (generic function), 309
gadget-command-setter (generic function), 309

API Index 479
DUIM Reference Documentation, Release 1.0

gadget-value-range-setter (generic function), 328
gadget-value-setter (generic function), 329
gadget-value-type (generic function), 329
gadget-x-alignment (generic function), 330
gadget-y-alignment (generic function), 330
gadget? (generic function), 306
gesture-button (generic function), 159
gesture-keysym (generic function), 159
gesture-modifier-state (generic function), 159
gesture-spec-equal (function), 160
get-clipboard-data-as (generic function), 160
get-default-background (generic function), 160
get-default-foreground (generic function), 161
get-default-text-style (generic function), 161
hearts-stipple (constant), 91
horizontal-hatch (constant), 91
hyper-key (constant), 163
iconify-frame (generic function), 454
identity-transform? (generic function), 22
image-depth (generic function), 92
image-height (generic function), 92
image-width (generic function), 92
image? (generic function), 91
ink? (generic function), 93
invertible-transform (generic function), 22
invertible-transform? (generic function), 22
item-object (generic function), 331
key-press-event (class), 164
key-release-event (class), 164
keyboard-event (class), 163
keyboard-gesture (class), 164

large-coordinate (constant), 23
left-button (constant), 165
label (class), 332
layout (class), 263
leaf-pane (class), 265
line (class), 53
list-box (class), 333
list-control-view (type), 336
list-control (class), 333
list-item (class), 338
labelling (macro), 332
layout-border (generic function), 264
layout-border-setter (generic function), 264
layout-equalize-heights? (generic function), 264
layout-equalize-widths? (generic function), 265
layout-frame (generic function), 454
line-end-point (generic function), 54
line-end-position (generic function), 54
line-start-point (generic function), 54
line-start-position (generic function), 54
line-to (generic function), 226
line? (generic function), 53
list-control-icon-function (generic function), 335
list-control-icon-function-setter (generic function), 335
list-control-view (generic function), 336
list-control-view-setter (generic function), 337
lower-frame (generic function), 455
lower-sheet (generic function), 165
magenta (constant), 93
meta-key (constant), 175
middle-button (constant), 175
modifier-keys (constant), 177
medium (class), 166
menu-bar (class), 341
menu-box (class), 341
menu-button (class), 342
menu (class), 340
multiple-child-composite-pane (class), 266
duim-dcs:duim-dcs:make (generic function), 94
duim-layouts:duim-layouts:make (<space-requirement>) (method), 265
duim:duim-frames:make (<frame>) (method), 455
make-3-point-transform (function), 23
make-bounding-box (function), 24
make-color-for-contrasting-color (generic function), 94
make-contrasting-colors (function), 95
make-contrasting-dash-patterns (function), 95
make-device-font (function), 96
make-ellipse (function), 55
make-elliptical-arc (function), 56
make-frame-manager (generic function), 165
make-gray-color (function), 96
make-ihs-color (function), 97
make-item (generic function), 338
make-line (function), 57
make-menu-from-command-table-menu (generic function), 457
make-menu-from-items (generic function), 339
make-menus-from-command-table (generic function), 458
make-modifier-state (function), 166
make-node (generic function), 339
make-palette (generic function), 97
make-pattern (function), 166
make-pixmap (generic function), 248
make-point (function), 24
make-polygon (function), 57
make-polyline (function), 58
make-rectangle (function), 58
make-reflection-transform (function), 25
make-rgb-color (function), 98
make-rotation-transform (function), 26
make-scaling-transform (function), 27
make-stencil (function), 98
make-text-style (function), 99
make-transform (function), 27
make-translation-transform (function), 28
medium-background (generic function), 168
medium-background-setter (generic function), 168
medium-brush (generic function), 169
medium-brush-setter (generic function), 169
medium-clipping-region (generic function), 169
medium-clipping-region-setter (generic function), 170
medium-default-text-style (generic function), 170
medium-default-text-style-setter (generic function), 170
medium-text-style-setter (generic function), 174
medium-transform (generic function), 174
medium-transform-setter (generic function), 175
medium? (generic function), 168
menu-item-accelerator (generic function), 458
menu-item-mnemonic (generic function), 458
menu-item-name (generic function), 459
menu-item-options (generic function), 459
menu-item-type (generic function), 459
menu-item-value (generic function), 460
menu-owner (generic function), 343
merge-text-styles (generic function), 99
modifier-key-index (function), 176
modifier-key-index-name (function), 176
move-to (generic function), 226
move-to-next-page (generic function), 460
move-to-previous-page (generic function), 460

N

$nowhere (constant), 29
<null-pane> (class), 266
node-children (generic function), 343
node-children-setter (generic function), 343
node-expanded? (generic function), 344
node-object (generic function), 344
node-parents (generic function), 344
node-state (generic function), 345
note-progress (generic function), 460
notify-user (generic function), 177
noting-progress (macro), 461

O

$option-key (constant), 179
<option-box> (class), 345
open-clipboard (function), 178

P

$parquet-stipple (constant), 101
:pointer-buttons (constant), 181
*progress-note* (variable), 461
?page> (class), 345
<palette-full> (class), 100
<palette> (class), 100
<password-field> (class), 346
<path> (class), 29
<pattern> (class), 101
<pen> (class), 101
<pinboard-layout> (class), 267
<pixmap-medium> (class), 250
<pixmap> (class), 249
<point> (class), 29
<pointer-boundary-event> (class), 180
<pointer-button-event> (class), 180
<pointer-drag-event> (class), 182
<pointer-enter-event> (class), 183
<pointer-event> (class), 183
<pointer-exit-event> (class), 183
<pointer-gesture> (class), 183
<pointer-motion-event> (class), 184
<pointer> (class), 179
<polygon> (class), 59
<polyline> (class), 61
<port-terminated-event> (class), 188
<port> (class), 185
<progress-bar> (class), 346
<property-frame> (class), 462
<property-page> (class), 463
<push-box> (class), 347
<push-button> (class), 348
<push-menu-box> (class), 348
<push-menu-button> (class), 349
palette? (generic function), 100
pane-display-function (generic function), 266
pane-layout (generic function), 267
path? (generic function), 29
pattern? (generic function), 101
pen-cap-shape (generic function), 103
pen-dashes (generic function), 103
pen-joint-shape (generic function), 104
pen-units (generic function), 104
pen-width (generic function), 104
pen? (generic function), 102
pixmap? (generic function), 250
point-position (generic function), 30
point-x (generic function), 30
point-y (generic function), 31
point? (generic function), 30
pointer-button-state (generic function), 181
pointer-cursor (generic function), 182
pointer-cursor-setter (generic function), 182
pointer-position (generic function), 184
pointer-sheet (generic function), 184
pointer? (generic function), 180
polygon-coordinates (generic function), 60
polygon-points (generic function), 60
polygon? (generic function), 60
polyline-closed? (generic function), 62
polyline? (generic function), 62
port (generic function), 185
port-modifier-state (generic function), 186
port-name (generic function), 187
port-pointer (generic function), 187
port-server-path (generic function), 187
port-type (generic function), 188
port? (generic function), 186

Q
queue-event (generic function), 188
queue-repaint (generic function), 188

R
$red (constant), 106
$right-button (constant), 190
<radio-box> (class), 349
<radio-button> (class), 350
<radio-menu-box> (class), 350
<radio-menu-button> (class), 351
<rectangle> (class), 62
<reflection-underspecified> (class), 32
<region-set> (class), 35
<region> (class), 32
<row-layout> (class), 268
raise-frame (generic function), 463
raise-sheet (generic function), 189
read-image (generic function), 105
read-image-as (generic function), 105
rectangle-edges (generic function), 64
rectangle-height (generic function), 65
rectangle-max-point (generic function), 65
rectangle-max-position (generic function), 66
rectangle-min-point (generic function), 66
rectangle-min-position (generic function), 67
rectangle-size (generic function), 67
rectangle-width (generic function), 68
rectangle? (generic function), 64
rectilinear-transform? (generic function), 31
redo-command (generic function), 464
reflection-transform? (generic function), 31
region-contains-position? (generic function), 33
region-contains-region? (generic function), 33
region-difference (generic function), 34
region-empty? (generic function), 34
region-equal (generic function), 34
region-intersection (generic function), 35
region-intersects-region? (generic function), 35
region-set-function (generic function), 36
region-set-regions (generic function), 36
region-set? (generic function), 36
region-union (generic function), 37
region? (generic function), 33
relayout-children (generic function), 268
relayout-parent (generic function), 268
remove-child (generic function), 189
remove-colors (generic function), 106
remove-column (generic function), 352
remove-command (generic function), 464
remove-command-table (function), 465
remove-command-table-menu-item (generic function), 465
remove-item (generic function), 352
remove-node (generic function), 352
repaint-sheet (generic function), 189
replace-child (generic function), 190
restore-clipping-region (generic function), 251
rigid-transform? (generic function), 37
$shift-key (constant), 206
$smallest-coordinate (constant), 39
$solid- pen (constant), 106
$super-key (constant), 206
<scroll-bar> (class), 353
<separator> (class), 355
<sheet-event> (class), 199
<sheet> (class), 192
<simple-command> (class), 466
<simple-frame> (class), 466
<simple-pane> (class), 270
<simple-undoable-command> (class), 467
<single-child-composite-pane> (class), 270
<singular-transform> (class), 39
<slider> (class), 356
<space-requirement> (class), 270
<spacing> (class), 357
<spin-box> (class), 358
<splitter> (class), 359
<stack-layout> (class), 274
<status-bar> (class), 360
<stencil> (class), 106
scaling-transform? (generic function), 38
scroll-position (generic function), 354
scrolling (macro), 354
set-box-edges (generic function), 38
set-box-position (generic function), 38
set-box-size (generic function), 38
set-caret-position (generic function), 190
set-frame-position (generic function), 465
set-frame-size (generic function), 465
set-pointer-position (generic function), 191
set-scroll-position (generic function), 355
set-sheet-edges (generic function), 191
set-sheet-position (generic function), 191
set-sheet-size (generic function), 192
sheet-ancestor? (generic function), 197
sheet-child (generic function), 198
sheet-child-setter (generic function), 199
sheet-children (generic function), 198
sheet-children-setter (generic function), 198
sheet-edges (generic function), 199
sheet-event-mask (generic function), 200
sheet-event-mask-setter (generic function), 200
sheet-event-queue (generic function), 200
sheet-frame (generic function), 201
sheet-mapped? (generic function), 201
sheet-mapped?-setter (generic function), 201
sheet-medium (generic function), 201
sheet-parent (generic function), 202
sheet-parent-setter (generic function), 202
sheet-pointer-cursor (generic function), 202
sheet-pointer-cursor-setter (generic function), 203
sheet-position (generic function), 203
sheet-region (generic function), 203
sheet-region-setter (generic function), 204
sheet-size (generic function), 204
sheet-state (generic function), 204
sheet-text-cursor (generic function), 205
sheet-transform (generic function), 205
sheet-transform-setter (generic function), 205
sheet-viewport (generic function), 356
sheet-viewport-region (generic function), 356
sheet-withdrawn? (generic function), 206
sheet? (generic function), 197
space-requirement-height (generic function), 272
space-requirement-max-height (generic function), 272
space-requirement-max-width (generic function), 273
space-requirement-min-height (generic function), 273
space-requirement-min-width (generic function), 273
space-requirement-width (generic function), 274
space-requirement? (generic function), 272
splitter-split-bar-moved-callback (generic function), 359
splitter-split-bar-moved-callback-setter (generic function), 359
splitter-split-box-callback (generic function), 359
splitter-split-box-callback-setter (generic function), 360
stack-layout-mapped-page (generic function), 275
stack-layout-mapped-page-setter (generic function), 275
start-dialog (generic function), 467
start-frame (generic function), 468
start-path (generic function), 225
status-bar-label-pane (generic function), 362

API Index 483
status-bar-progress-bar (generic function), 362
stencil? (generic function), 107
stroke-path (generic function), 226
synchronize-display (generic function), 207

T
tiles-stipple (constant), 113
<tab-control-page> (class), 366
<tab-control> (class), 363
<table-column> (class), 368
<table-control-view> (type), 370
<table-control> (class), 368
<table-item> (class), 371
<table-layout> (class), 276
<text-editor> (class), 372
<text-field> (class), 372
<text-gadget> (class), 373
<text-style> (class), 107
timer-event (class), 209
tool-bar (class), 374
top-level-sheet (class), 279
transform-error (class), 42
transform-underspecified (class), 43
<tree-control> (class), 375
tree-node (class), 381
tab-control-current-page (generic function), 364
tab-control-current-page-setter (generic function), 364
tab-control-labels (generic function), 365
tab-control-pages (generic function), 366
tab-control-pages-setter (generic function), 367
<table-contents> (generic function), 275
<table-contents-setter> (generic function), 275
<tab-control-view> (generic function), 371
<tab-control-view-setter> (generic function), 371
tabling (macro), 278
text-size (generic function), 207
text-style-components (generic function), 109
text-style-family (generic function), 110
text-style-mapping (generic function), 208
text-style-mapping-exists? (generic function), 208
text-style-mapping-setter (generic function), 209
text-style-size (generic function), 110
text-style-slant (generic function), 111
text-style-strikeout? (generic function), 111
text-style-underline? (generic function), 112
text-style-weight (generic function), 112

text-style? (generic function), 108
top-level-sheet (generic function), 209
transform-angles (generic function), 41
transform-box (generic function), 41
transform-distance (generic function), 41
transform-position (generic function), 42
transform-region (generic function), 42
transform? (generic function), 40
translation-transform? (generic function), 43
<tree-control-children-generator> (generic function), 378
<tree-control-children-generator-setter> (generic function), 378
<tree-control-children-predicate> (generic function), 377
<tree-control-children-predicate-setter> (generic function), 377
<tree-control-icon-function> (generic function), 378
<tree-control-initial-depth> (generic function), 379
<tree-control-initial-depth-setter> (generic function), 379
<tree-control-roots> (generic function), 380
<tree-control-roots-setter> (generic function), 380

U
<user-command-table> (variable), 469
<undefined-text-style-mapping> (class), 209
undo-command (generic function), 468
untransform-angles (generic function), 43
untransform-box (generic function), 44
untransform-distance (generic function), 44
untransform-position (generic function), 45
untransform-region (generic function), 45
update-gadget (generic function), 381

V
$vertical-hatch (constant), 113
<value-gadget> (class), 382
<value-range-gadget> (class), 382
<viewport> (class), 383
vertically (macro), 279
viewport-region (generic function), 384
viewport? (generic function), 384

W
$white (constant), 113
<window-configuration-event> (class), 210
<window-event> (class), 210
<window-repaint-event> (class), 210
<wizard-frame> (class), 469
<wizard-page> (class), 471
with-border (macro), 384
with-brush (macro), 211
with-clipboard (macro), 211
with-clipping-region (macro), 212
with-cursor-visible (macro), 212
with-drawing-options (macro), 212
with-frame-manager (macro), 213
with-identity-transform (macro), 213
with-output-to-pixmap (macro), 251
with-pen (macro), 214
with-pointer-grabbed (macro), 214
with-rotation (macro), 214
with-scaling (macro), 215
with-sheet-medium (macro), 215
with-spacing (macro), 385
with-text-style (macro), 216
with-transform (macro), 216
with-translation (macro), 217
withdraw-sheet (generic function), 213
write-image (generic function), 113

X

$xor-brush (constant), 113

Y

$yellow (constant), 113
Symbols
*global-command-table*, 453
*progress-note*, 461
*user-command-table*, 469
=, 70, 120
  =(<command>), 390
  =(<region>), 10
  =(<transform>), 10
$alt-key, 121
$background, 71
$black, 71
$blue, 71
$boole-1, 72
$boole-2, 72
$boole-and, 72
$boole-andc1, 73
$boole-andc2, 73
$boole-c1, 72
$boole-c2, 72
$boole-clr, 71
$boole-eqv, 72
$boole-ior, 72
$boole-nand, 73
$boole-nor, 72
$boole-orc1, 73
$boole-orc2, 73
$boole-set, 71
$boole-xor, 72
$boole-nand, 73
$boole-nor, 73
$boole-orc1, 73
$boole-orc2, 73
$boole-set, 71
$boole-xor, 72
$bricks-stipple, 73
$control-key, 136
$cross-hatch, 82
$cyan, 82
$dash-dot-dot-pen, 82
$dash-dot-pen, 82
$ashed-pen, 83
$diagonal-hatch-down, 85
$diagonal-hatch-up, 85
$dotted-pen, 86
$everywhere, 21
$sfill, 261
$foreground, 90
$green, 90
$hearts-stipple, 91
$horizontal-hatch, 91
$hyper-key, 163
$identity-transform, 21
$largest-coordinate, 23
$left-button, 165
$magenta, 93
$meta-key, 175
$middle-button, 175
$modifier-keys, 177
$nowhere, 29
$option-key, 179
$parquet-stipple, 101
$pointer-stipple, 181
$red, 106
$right-button, 190
$shift-key, 206
$smallest-coordinate, 39
$solid-pen, 106
$super-key, 206
$stiles-stipple, 113
$svertical-hatch, 113
$white, 113
$ xor-brush, 113
$yellow, 113
<action-gadget>, 291
<application-exited-event>, 393
<area>, 11
<basic-user-pane>, 255
<border>, 293
<bounding-box>, 11
<brush>, 73
<button-box>, 296
<button-press-event>, 123
<button-release-event>, 123
<button>, 294
<caret>, 123
<check-box>, 297
<check-button>, 298
<check-menu-box>, 298
<check-menu-button>, 299
<clipboard>, 134
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection-gadget</td>
<td>299</td>
</tr>
<tr>
<td>color-not-found</td>
<td>80</td>
</tr>
<tr>
<td>color</td>
<td>77</td>
</tr>
<tr>
<td>column-layout</td>
<td>256</td>
</tr>
<tr>
<td>combo-box</td>
<td>301</td>
</tr>
<tr>
<td>command-table-menu-item</td>
<td>400</td>
</tr>
<tr>
<td>command-table</td>
<td>398</td>
</tr>
<tr>
<td>command</td>
<td>395</td>
</tr>
<tr>
<td>cursor</td>
<td>137</td>
</tr>
<tr>
<td>device-event</td>
<td>138</td>
</tr>
<tr>
<td>device-font</td>
<td>85</td>
</tr>
<tr>
<td>dialog-frame</td>
<td>420</td>
</tr>
<tr>
<td>display</td>
<td>138</td>
</tr>
<tr>
<td>double-click-event</td>
<td>146</td>
</tr>
<tr>
<td>drawing-pane</td>
<td>261</td>
</tr>
<tr>
<td>ellipse</td>
<td>48</td>
</tr>
<tr>
<td>elliptical-arc</td>
<td>51</td>
</tr>
<tr>
<td>event</td>
<td>148</td>
</tr>
<tr>
<td>fixed-layout</td>
<td>261</td>
</tr>
<tr>
<td>frame-created-event</td>
<td>440</td>
</tr>
<tr>
<td>frame-destroyed-event</td>
<td>440</td>
</tr>
<tr>
<td>frame-event</td>
<td>156</td>
</tr>
<tr>
<td>frame-exit-event</td>
<td>441</td>
</tr>
<tr>
<td>frame-exited-event</td>
<td>441</td>
</tr>
<tr>
<td>frame-focus-event</td>
<td>442</td>
</tr>
<tr>
<td>frame-manager</td>
<td>156</td>
</tr>
<tr>
<td>frame-mapped-event</td>
<td>445</td>
</tr>
<tr>
<td>frame-unmapped-event</td>
<td>453</td>
</tr>
<tr>
<td>frame</td>
<td>434</td>
</tr>
<tr>
<td>gadget</td>
<td>304</td>
</tr>
<tr>
<td>gesture</td>
<td>158</td>
</tr>
<tr>
<td>grid-layout</td>
<td>262</td>
</tr>
<tr>
<td>group-box</td>
<td>331</td>
</tr>
<tr>
<td>image</td>
<td>91</td>
</tr>
<tr>
<td>ink</td>
<td>93</td>
</tr>
<tr>
<td>key-press-event</td>
<td>164</td>
</tr>
<tr>
<td>key-release-event</td>
<td>164</td>
</tr>
<tr>
<td>keyboard-event</td>
<td>163</td>
</tr>
<tr>
<td>keyboard-gesture</td>
<td>164</td>
</tr>
<tr>
<td>label</td>
<td>332</td>
</tr>
<tr>
<td>layout</td>
<td>263</td>
</tr>
<tr>
<td>leaf-pane</td>
<td>265</td>
</tr>
<tr>
<td>line</td>
<td>53</td>
</tr>
<tr>
<td>list-box</td>
<td>333</td>
</tr>
<tr>
<td>list-control-view</td>
<td>336</td>
</tr>
<tr>
<td>list-control</td>
<td>333</td>
</tr>
<tr>
<td>list-item</td>
<td>338</td>
</tr>
<tr>
<td>medium</td>
<td>166</td>
</tr>
<tr>
<td>menu-bar</td>
<td>341</td>
</tr>
<tr>
<td>menu-box</td>
<td>341</td>
</tr>
<tr>
<td>menu-button</td>
<td>342</td>
</tr>
<tr>
<td>menu</td>
<td>340</td>
</tr>
<tr>
<td>multiple-child-composite-pane</td>
<td>266</td>
</tr>
<tr>
<td>null-pane</td>
<td>266</td>
</tr>
<tr>
<td>option-box</td>
<td>345</td>
</tr>
<tr>
<td>page</td>
<td>345</td>
</tr>
<tr>
<td>palette-full</td>
<td>100</td>
</tr>
<tr>
<td>palette</td>
<td>100</td>
</tr>
<tr>
<td>password-field</td>
<td>346</td>
</tr>
<tr>
<td>path</td>
<td>29</td>
</tr>
<tr>
<td>pattern</td>
<td>101</td>
</tr>
<tr>
<td>pen</td>
<td>101</td>
</tr>
<tr>
<td>pinboard-layout</td>
<td>267</td>
</tr>
<tr>
<td>pixmap-medium</td>
<td>250</td>
</tr>
<tr>
<td>pixmap</td>
<td>249</td>
</tr>
<tr>
<td>point</td>
<td>29</td>
</tr>
<tr>
<td>pointer-boundary-event</td>
<td>180</td>
</tr>
<tr>
<td>pointer-button-event</td>
<td>180</td>
</tr>
<tr>
<td>pointer-drag-event</td>
<td>182</td>
</tr>
<tr>
<td>pointer-enter-event</td>
<td>183</td>
</tr>
<tr>
<td>pointer-event</td>
<td>183</td>
</tr>
<tr>
<td>pointer-exit-event</td>
<td>183</td>
</tr>
<tr>
<td>pointer-gesture</td>
<td>183</td>
</tr>
<tr>
<td>pointer-motion-event</td>
<td>184</td>
</tr>
<tr>
<td>pointer</td>
<td>179</td>
</tr>
<tr>
<td>polygon</td>
<td>59</td>
</tr>
<tr>
<td>polyline</td>
<td>61</td>
</tr>
<tr>
<td>port-terminated-event</td>
<td>188</td>
</tr>
<tr>
<td>port</td>
<td>185</td>
</tr>
<tr>
<td>progress-bar</td>
<td>346</td>
</tr>
<tr>
<td>property-frame</td>
<td>462</td>
</tr>
<tr>
<td>property-page</td>
<td>463</td>
</tr>
<tr>
<td>push-box</td>
<td>347</td>
</tr>
<tr>
<td>push-button</td>
<td>348</td>
</tr>
<tr>
<td>push-menu-box</td>
<td>348</td>
</tr>
<tr>
<td>push-menu-button</td>
<td>349</td>
</tr>
<tr>
<td>push-menu-button</td>
<td>349</td>
</tr>
<tr>
<td>radio-box</td>
<td>349</td>
</tr>
<tr>
<td>radio-button</td>
<td>350</td>
</tr>
<tr>
<td>radio-menu-box</td>
<td>350</td>
</tr>
<tr>
<td>radio-menu-button</td>
<td>351</td>
</tr>
<tr>
<td>rectangle</td>
<td>62</td>
</tr>
<tr>
<td>reflection-underspecified</td>
<td>32</td>
</tr>
<tr>
<td>region-set</td>
<td>35</td>
</tr>
<tr>
<td>region</td>
<td>32</td>
</tr>
<tr>
<td>row-layout</td>
<td>268</td>
</tr>
<tr>
<td>scroll-bar</td>
<td>353</td>
</tr>
<tr>
<td>separator</td>
<td>355</td>
</tr>
<tr>
<td>sheet-event</td>
<td>199</td>
</tr>
<tr>
<td>sheet</td>
<td>192</td>
</tr>
<tr>
<td>simple-command</td>
<td>466</td>
</tr>
<tr>
<td>simple-frame</td>
<td>466</td>
</tr>
<tr>
<td>simple-pane</td>
<td>270</td>
</tr>
<tr>
<td>simple-undoable-command</td>
<td>467</td>
</tr>
<tr>
<td>single-child-composite-pane</td>
<td>270</td>
</tr>
<tr>
<td>singular-transform</td>
<td>39</td>
</tr>
<tr>
<td>slider</td>
<td>356</td>
</tr>
<tr>
<td>space-requirement</td>
<td>270</td>
</tr>
<tr>
<td>spacing</td>
<td>357</td>
</tr>
</tbody>
</table>
<spin-box>, 358
<spliter>, 359
<stack-layout>, 274
<status-bar>, 360
<stencil>, 106
<tab-control-page>, 366
<tab-control>, 363
<table-column>, 368
<table-control-view>, 370
<table-control>, 368
<table-item>, 371
<table-layout>, 276
<text-editor>, 372
<text-field>, 372
<text-gadget>, 373
<text-style>, 107
<timer-event>, 209
<tool-bar>, 374
<top-level-sheet>, 279
<transform-error>, 42
<transform-underspecified>, 43
<transform>, 39
<tree-control>, 375
<tree-node>, 381
<undefined-text-style-mapping>, 209
:value-gadget>, 382
:value-range-gadget>, 382
<viewport>, 383
>window-configuration-event>, 210
>window-event>, 210
>window-repaint-event>, 210
<wizard-frame>, 469
<wizard-page>, 471

A
abort-path, 226, 227
activate-gadget, 292
add-child, 120
add-clipboard-data, 121
add-clipboard-data-as, 121
add-colors, 71
add-column, 292
add-command, 390
add-command-table-menu-item, 391
add-item, 292
add-node, 293
allocate-space, 255
apply-in-frame, 393
arc-to, 226, 227
area?, 11

B
beep, 122
boundary-event-kind, 122
bounding-box, 12
bounding-box?, 12
box-bottom, 13
box-edges, 13
box-height, 14
box-left, 14
box-position, 15
box-right, 15
box-size, 15
box-top, 16
box-width, 16
brush-background, 75
brush-fill-rule, 75
brush-fill-style, 75
brush-foreground, 75
brush-mode, 76
brush-stipple, 76
brush-stretch-mode, 76
brush-tile, 76
brush-ts-x, 77
brush-ts-y, 77
brush?, 74
button-index, 122
button-index-name, 122

C
call-in-frame, 394
cancel-dialog, 394
caret-position, 124
caret-sheet, 124
caret-size, 125
caret-visible?, 125
caret-visible?-setter, 125
child-containing-position, 125
children-overlapping-region, 126
choose-color, 126
choose-directory, 128
choose-file, 129
choose-from-dialog, 131
choose-from-menu, 132
choose-text-style, 133
clear-box, 134
clear-clipboard, 134
clear-progress-note, 395
clipboard-data-available?, 135
clipboard-owner, 136
clipboard-sheet, 135
close-clipboard, 136
close-path, 225, 229
color-ihs, 79
color-luminosity, 80
color-palette?, 80
color-rgb, 81
color?, 79
command-arguments, 396
command-enabled?, 396
command-enabled?-setter, 397
command-function, 397
command-table-accelerators, 399
command-table-commands, 399
command-table-menu, 400
command-table-name, 401
command-table?, 399
command-undoable?, 401
command?, 396
complete-from-generator, 402
complete-from-sequence, 403
compose-rotation-with-transform, 16
compose-scaling-with-transform, 17
compose-space, 258
compose-transform-with-rotation, 18
compose-transform-with-scaling, 18
compose-transform-with-translation, 19
compose-transforms, 17
compose-translation-with-transform, 19
compute-next-page, 404
compute-previous-page, 404
contain, 405
contract-node, 302
contrasting-colors-limit, 81
contrasting-dash-patterns-limit, 81
copy-area, 229
copy-from-pixmap, 230
copy-to-pixmap, 230
current-frame, 406
current-pane, 258
cursor?, 137
curve-to, 226, 231

D
default-background, 83
default-background-setter, 83
default-foreground, 84
default-foreground-setter, 84
default-port, 137
default-port-setter, 137
default-text-style, 84
default-text-style-setter, 85
define command-table, 406
define frame, 408
define-pane, 259
deiconify-frame, 410
destroy-frame, 411
destroy-pixmap, 232
destroy-port, 138
destroy-sheet, 138
dialog-apply-button, 411
dialog-apply-button-setter, 412
dialog-apply-callback, 412
dialog-back-button, 413
dialog-back-button-setter, 413
dialog-back-callback, 413
dialog-cancel-button, 414
dialog-cancel-button-setter, 414
dialog-cancel-callback, 415
dialog-cancel-callback-setter, 415
dialog-current-page, 416
dialog-current-page-setter, 416
dialog-exit-button, 416
dialog-exit-button-setter, 417
dialog-exit-callback, 418
dialog-exit-callback-setter, 418
dialog-exit-enabled?, 419
dialog-exit-enabled?-setter, 419
dialog-help-button, 422
dialog-help-button-setter, 423
dialog-help-callback, 424
dialog-next-button, 424
dialog-next-button-setter, 425
dialog-next-callback, 425
dialog-next-enabled?, 426
dialog-next-enabled?-setter, 426
dialog-next-page, 427
dialog-next-page-setter, 427
dialog-page-changed-callback, 428
dialog-page-changed-callback-setter, 428
dialog-page-complete?, 428
dialog-page-complete?-setter, 429
dialog-pages, 429
dialog-pages-setter, 430
dialog-previous-page, 430
dialog-previous-page-setter, 431
display, 139
display-depth, 140
display-height, 140
display-menu, 302
display-mm-height, 141
display-mm-width, 141
display-orientation, 142
display-pixel-height, 142
display-pixel-width, 143
display-pixels-per-point, 142
display-progress-note, 431
display-units, 143
display-width, 143
display?, 140
do-allocate-space, 259
do-children-containing-position, 144
do-children-overlapping-region, 144
do-compose-space, 260
do-coordinates, 20
do-displays, 145
do-endpoint-coordinates, 20
do-frames, 145
do-polygon-coordinates, 48
do-polygon-segments, 48
do-ports, 145
do-regions, 20
do-sheet-children, 145
do-sheet-tree, 145
do-with-drawing-options, 146
do-with-output-to-pixmap, 232
do-with-pointer-grabbed, 146
do-with-sheet-medium, 147
do-with-text-style, 147
do-with-transform, 147
draw-arrow, 232
draw-bezier-curve, 234
draw-circle, 234
draw-design, 48
draw-ellipse, 235
draw-image, 236
draw-line, 237
draw-lines, 238
draw-oval, 238
draw-pixmap, 239
draw-point, 240
draw-points, 241
draw-polygon, 241
draw-rectangle, 242
draw-rectangles, 243
draw-regular-polygon, 244
draw-text, 245
draw-triangle, 246

E
ellipse-center-point, 50
ellipse-center-position, 50
ellipse-end-angle, 50
ellipse-radii, 51
ellipse-start-angle, 51
ellipse?, 49
elliptical-arc?, 52
end-path, 225, 247
even-scaling-transform?, 21
event-button, 148
event-character, 149
event-destroy-frame?, 431
event-key-name, 149
event-matches-gesture?, 150
event-modifier-state, 150
event-pointer, 150
event-region, 151
event-sheet, 151
event-status-code, 432
event-x, 151
event-y, 152
event?, 148
execute-command, 432
exit-dialog, 432
exit-frame, 433
expand-node, 302

F
fill-path, 226, 247
find-color, 86
find-display, 152
find-frame, 434
find-frame-manager, 152
find-item, 303
find-node, 303
find-port, 153
fix-coordinate, 21
fixed-width-font?, 153
font-ascent, 153
font-descent, 154
font-height, 154
font-metrics, 155
font-width, 155
force-display, 156
frame-accelerators, 438
frame-accelerators-setter, 438
frame-can-exit?, 439
frame-command-table, 439
frame-command-table-setter, 439
frame-default-button, 440
frame-default-button-setter, 440
frame-event-queue, 441
frame-fixed-height?, 442
frame-fixed-width?, 442
frame-icon, 443
frame-icon-setter, 443
frame-input-focus, 443
frame-input-focus-setter, 444
frame-layout, 444
frame-layout-setter, 444
frame-manager, 157
frame-manager-frames, 158
frame-manager-palette, 158
frame-manager-palette-setter, 158
frame-manager?, 157
frame-mapped?, 444
frame-mapped?-setter, 445
frame-menu-bar, 446
frame-menu-bar-setter, 446
frame-mode, 447
frame-owner, 447
frame-palette, 447
frame-palette-setter, 448
frame-position, 448

Index 491
frame-resizable?, 449
frame-size, 449
frame-state, 449
frame-status-bar, 450
frame-status-bar-setter, 450
frame-status-message, 450
frame-status-message-setter, 451
frame-thread, 451
frame-title, 451
frame-title-setter, 452
frame-tool-bar, 452
frame-tool-bar-setter, 452
frame-top-level, 453
frame?, 438
fully-merged-text-style?, 90

gadget-accelerator, 306
gadget-accelerator-setter, 306
gadget-activate-callback, 307
gadget-activate-callback-setter, 307
gadget-client, 308
gadget-client-setter, 308
gadget-command, 309
gadget-command-setter, 309
gadget-default?, 309
gadget-default?-setter, 310
gadget-documentation, 310
gadget-documentation-setter, 311
gadget-enabled?, 311
gadget-enabled?-setter, 312
gadget-id, 312
gadget-id-setter, 313
gadget-items, 314
gadget-items-setter, 314
gadget-key-press-callback, 315
gadget-key-press-callback-setter, 315
gadget-label, 316
gadget-label-key, 316
gadget-label-setter, 317
gadget-mnemonic, 317
gadget-mnemonic-setter, 318
gadget-orientation, 318
gadget-popup-menu-callback, 318
gadget-popup-menu-callback-setter, 319
gadget-ratios, 319
gadget-ratios-setter, 319
gadget-read-only?, 320
gadget-scrolling-horizontally?, 320
gadget-scrolling-vertically?, 320
gadget-selection, 320
gadget-selection-mode, 321
gadget-selection-setter, 322
gadget-slug-size, 322
gadget-slug-size-setter, 323
gadget-test, 323
gadget-text, 324
gadget-text-setter, 324
gadget-value, 325
gadget-value-changed-callback, 326
gadget-value-changed-callback-setter, 326
gadget-value-changing-callback, 326
gadget-value-changing-callback-setter, 327
gadget-value-key, 327
gadget-value-range, 328
gadget-value-range-setter, 328
gadget-value-setter, 329
gadget-value-type, 329
gadget-x-alignment, 330
gadget-y-alignment, 330
gadget?, 306
gesture-button, 159
gesture-keysym, 159
gesture-modifier-state, 159
gesture-spec-equal, 160
get-clipboard-data-as, 160
get-default-background, 160
get-default-foreground, 161
get-default-text-style, 161

handle-event, 162
handle-repaint, 162
horizontally, 262

iconify-frame, 454
identity-transform?, 22
image-dep, 92
image-height, 92
image-width, 92
image?, 91
ink?, 93
invert-transform, 22
invertible-transform?, 22
item-object, 331

labelling, 332
layout-border, 264
layout-border-setter, 264
layout-equalize-heights?, 264
layout-equalize-widths?, 265
layout-frame, 454
line-end-point, 54
line-end-position, 54
line-start-point, 54
line-start-position, 54
line-to, 226, 248
line?, 53
list-control-icon-function, 335
list-control-icon-function-setter, 335
list-control-view, 336
list-control-view-setter, 337
lower-frame, 455
lower-sheet, 165

M
make, 94
  make(<frame>), 455
  make(<space-requirement>), 265
make-3-point-transform, 23
make-bounding-box, 24
make-color-for-contrasting-color, 94
make-contrasting-colors, 95
make-contrasting-dash-patterns, 95
make-device-font, 96
make-ellipse, 55
make-elliptical-arc, 56
make-frame-manager, 165
make-gray-color, 96
make-ihs-color, 97
make-item, 338
make-line, 57
make-menu-from-command-table-menu, 457
make-menu-from-items, 339
make-menus-from-command-table, 458
make-modifier-state, 166
make-node, 339
make-palette, 97
make-pane, 166
make-pattern, 97
make-pixmap, 248
make-point, 24
make-polygon, 57
make-polyline, 58
make-rectangle, 58
make-reflection-transform, 25
make-rgb-color, 98
make-rotation-transform, 26
make-scaling-transform, 27
make-stencil, 98
make-text-style, 99
make-transform, 27
make-translation-transform, 28
medium-background, 168
medium-background-setter, 168
medium-brush, 169
medium-brush-setter, 169
medium-clipping-region, 169
medium-clipping-region-setter, 170
medium-default-text-style, 170
medium-default-text-style-setter, 170
medium-drawable, 171
medium-drawable-setter, 171
medium-foreground, 171
medium-foreground-setter, 172
medium-merged-text-style, 172
medium-pen, 172
medium-pen-setter, 173
medium-pixmap, 173
medium-pixmap-setter, 173
medium-sheet, 173
medium-text-style, 174
medium-text-style-setter, 174
medium-transform, 174
medium-transform-setter, 175
medium?, 168
menu-item-accelerator, 458
menu-item-nemonic, 458
menu-item-name, 459
menu-item-options, 459
menu-item-type, 459
menu-item-value, 460
menu-owner, 343
merge-text-styles, 99
modifier-key-index, 176
modifier-key-index-name, 176
move-to, 226, 249
move-to-next-page, 460
move-to-previous-page, 460

N
node-children, 343
node-children-setter, 343
node-expanded?, 344
node-object, 344
node-parents, 344
node-state, 345
note-progress, 460
notify-user, 177
noting-progress, 461

O
open-clipboard, 178

P
palette?, 100
pane-display-function, 266
pane-layout, 267
path?, 29
pattern?, 101
pen-cap-shape, 103
pen-dashes, 103
pen-joint-shape, 104
pen-units, 104

Index 493
pen-width, 104
pen?, 102
pixmap?, 250
point-position, 30
point-x, 30
point-y, 31
point?, 30
pointer-button-state, 181
pointer-cursor, 182
pointer-cursor-setter, 182
pointer-position, 184
pointer-sheet, 184
pointer?, 180
polygon-coordinates, 60
polygon-points, 60
polygon?, 60
polyline-closed?, 62
polyline?, 62
port, 185
port-modifier-state, 186
port-name, 187
port-pointer, 187
port-server-path, 187
port-type, 188
port?, 186

Q
queue-event, 188
queue-repaint, 188

R
raise-frame, 463
raise-sheet, 189
read-image, 105
read-image-as, 105
rectangle-edges, 64
rectangle-height, 65
rectangle-max-point, 65
rectangle-max-position, 66
rectangle-min-point, 66
rectangle-min-position, 67
rectangle-size, 67
rectangle-width, 68
rectangle?, 64
rectilinear-transform?, 31
redo-command, 464
reflection-transform?, 31
region-contains-position?, 33
region-contains-region?, 33
region-difference, 34
region-empty?, 34
region-equal, 34
region-intersection, 35
region-intersects-region?, 35
region-set-function, 36
region-set-regions, 36
region-set?, 36
region-union, 37
region?, 33
relayout-children, 268
relayout-parent, 268
remove-child, 189
remove-colors, 106
remove-column, 352
remove-command, 464
remove-command-table, 465
remove-command-table-menu-item, 465
remove-item, 352
remove-node, 352
repaint-sheet, 189
replace-child, 190
restore-clipping-region, 251
rigid-transform?, 37

S
scaling-transform?, 38
scroll-position, 354
scrolling, 354
set-box-edges, 38
set-box-position, 38
set-box-size, 38
set-caret-position, 190
set-frame-position, 465
set-frame-size, 465
set-pointer-position, 191
set-scroll-position, 355
set-sheet-edges, 191
set-sheet-position, 191
set-sheet-size, 192
sheet-ancestor?, 197
sheet-child, 198
sheet-child-setter, 199
sheet-children, 198
sheet-children-setter, 198
sheet-edges, 199
sheet-event-mask, 200
sheet-event-mask-setter, 200
sheet-event-queue, 200
sheet-frame, 201
sheet-mapped?, 201
sheet-mapped?-setter, 201
sheet-medium, 201
sheet-parent, 202
sheet-parent-setter, 202
sheet-pointer-cursor, 202
sheet-pointer-cursor-setter, 203
sheet-position, 203
sheet-region, 203
sheet-region-setter, 204
sheet-size, 204
sheet-state, 204
sheet-text-cursor, 205
sheet-transform, 205
sheet-transform-setter, 205
sheet-viewport, 356
sheet-viewport-region, 356
sheet-withdrawn?, 206
sheet?, 197
space-requirement-height, 272
space-requirement-max-height, 272
space-requirement-max-width, 273
space-requirement-min-height, 273
space-requirement-min-width, 273
space-requirement-width, 274
space-requirement?, 272
splitter-split-bar-moved-callback, 359
splitter-split-bar-moved-callback-setter, 359
splitter-split-box-callback, 359
splitter-split-box-callback-setter, 360
stack-layout-mapped-page, 275
stack-layout-mapped-page-setter, 275
start-dialog, 467
start-frame, 468
start-path, 225, 251
status-bar-label-pane, 362
status-bar-progress-bar, 362
stencil?, 107
stroke-path, 226, 251
synchronize-display, 207

tab-control-current-page, 364
tab-control-current-page-setter, 364
tab-control-labels, 365
tab-control-pages, 366
tab-control-pages-setter, 367
table-contents, 275
table-contents-setter, 275
table-control-view, 371
table-control-view-setter, 371
tablimg, 278
text-size, 207
text-style-components, 109
text-style-family, 110
text-style-mapping, 208
text-style-mapping-exists?, 208
text-style-mapping-setter, 209
text-style-size, 110
text-style-slant, 111
text-style-strikeout?, 111
text-style-underline?, 112
text-style-weight, 112
text-style?, 108
top-level-sheet, 209
transform-angles, 41
transform-box, 41
transform-distance, 41
transform-position, 42
transform-region, 42
transform?, 40
translation-transform?, 43
tree-control-children-generator, 378
tree-control-children-generator-setter, 378
tree-control-children-predicate, 377
tree-control-children-predicate-setter, 377
tree-control-icon-function, 378
tree-control-initial-depth, 379
tree-control-initial-depth-setter, 379
tree-control-roots, 380
tree-control-roots-setter, 380

U
undo-command, 468
untransform-angles, 43
untransform-box, 44
untransform-distance, 44
untransform-position, 45
untransform-region, 45
update-gadget, 381

V
vertically, 279
viewport, 384
viewport?, 384

W
with-border, 384
with-brush, 211
with-clipboard, 211
with-clipping-region, 212
with-cursor-visible, 212
with-drawing-options, 212
with-frame-manager, 213
with-identity-transform, 213
with-identity-transform?, 213
with-output-to-pixmap, 251
with-pen, 214
with-pointer-grabbed, 214
with-rotation, 214
with-scaling, 215
with-sheet-medium, 215
with-spacing, 385
with-text-style, 216
with-transform, 216
with-translation, 217
withdraw-sheet, 213
write-image, 113