ESR Consortium
MWT-1.0

Micro Widget Toolkit
Profile Specification

ESR011

Reference: ESR-SPE-011-MWT
Version: 1.0
Rev: D

DEFINITIONS

"ESR" means the Specification, including any modifications and upgrades, where these terms have been stated or referred to, and made available to You by ESR Consortium, including without limitation, texts, drawing, codes,and examples.

"ESR Consortium" means the non-profit entity, registered in France in accordance with the French law of 1901.

"You" means the legal entity or entities represented by the individual executing this Agreement.

READ ONLY RIGHTS

Subject to the terms and conditions contained herein, ESR Consortium grants to You a non-exclusive, non-transferable, worldwide, and royalty-free license to view and read the ESR solely for purposes of Your internal evaluation.

GENERAL TERMS

THIS DOCUMENTATION IS PROVIDED "AS IS", WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED.

THE READING OF THE ESR AND ALL CONSEQUENCES ARISING THEREOF IS YOUR SOLE RESPONSIBILITY. ESR CONSORTIUM SHALL NOT BE LIABLE TO YOU FOR ANY LOSS OR DAMAGE CAUSED BY, ARISING FROM, DIRECTLY OR INDIRECTLY, OR IN CONNECTION WITH THE ESR.

COPYRIGHT

ESR consortium does not have any right on this ESR. You are free to use this ESR to make any clean room implementations or derivative work as long as You doesn't not claim that Your work is compliant with the ESR. Compliance tests are available from the ESR Consortium.

MISCELLANEOUS

This Agreement shall be governed by, and interpreted in accordance with French Law. In no event shall this Agreement be construed against the drafter.

This Agreement contains the entire understanding between the parties concerning its subject matter and supersedes any other agreement or understanding, whether written or oral, which may exist or have existed between the parties on the subject matter hereof.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION.

ESR CONSORTIUM MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN ANY ESR PUBLICATION AT ANY TIME.

Trademarks

Java™ is Sun Microsystems' trademark for a technology for developing application software and deploying it in cross-platform, networked environments. When it is used in this documentation without adding the ™ symbol, it includes implementations of the technology by companies other than Sun.

Java™,all Java-based marks and all related logos are trademarks or registered trademarks of Sun Microsystems Inc, in the United States and other Countries.

Contents

Tables

Illustrations

1 PREFACE TO MWT 1.0 PROFILE, ESR011

This document defines the *Micro Widget Toolkit 1.0*, *MWT 1.0* profile , targeting Java 2 Platforms.

1.1 Who Should Use this Specification

This specification targets the following audiences:

- Platform Developers who want to build implementations that comply with the MWT profile specification;
- Application developers designing cross platform HMIs and using MWT;
- MicroUI, MIDP, AWT or SWT application developers;
- Java virtual machine providers deploying Java for Human Machine Interface devices.

1.2 How This Specification is Organized

This specification is organized as follow:

- **Introduction** is a short chapter explaining what MWT is, why it has been designed, and its main assets.
- **Basic Concepts**: aims at making the reader familiar with the fundamental MWT notions and vocabulary.
- **MWT Design** gives information needed to understand the MWT architecture.
- **MWT API Documentation** lists the MWT APIs as javadoc.

1.3 Comments

Your comments about MWT are welcome. Please send them by electronic mail to the following address: `comments @e-s-r.net` , with MWT in your subject line.

1.4 Glossary

- *HMI*: Human to Machine Interface
- *ESR*: Embedded Specification Request
- *JSR*: Java Specification Request
- *baremetal*: a Java virtual machine is said to be *baremetal* when it does not require an OS/RTOS to run. A baremetal Java virtual machine is in fact an OS/RTOS that also embeds a Java engine. The device boots directly in Java.

1.5 Related Literature

CLDC 1.1: Sun Microsystems, Inc., Connected Limited Device Configuration (JSR139), 2003, http://jcp.org/en/jsr/detail?id=139

MicroUI 1.4: ESR Consortium, Micro User Interface profile; ESR 002, 2010, http://www.e-s-r.net
B-ON 1.1: ESR Consortium, B-ON 1.1 Beyond CLDC : ESR 001, 2008, http://www.e-s-r.net

1.6 Document Conventions

In this document, references to methods of a Java class are written as `ClassName.methodName(args)`. This applies to both static and instance methods. Where the method is static this will be made clear in the accompanying text.

1.7 Implementation Notes

The MWT specification does not include any implementation details. MWT implementors are free to use whatever techniques they deem appropriate to implement the specification, with (or without) collaboration of any Java virtual machine provider. MWT experts have taken great care not to mention any special Java virtual machines, nor any of their special features, in order to encourage fair competing implementations.

2 INTRODUCTION

2.1 Requirements

The term MUST indicates that the associated definition is an absolute requirement, whereas MAY indicates that the item is optional. SHOULD indicates a highly recommended requirement.

2.1.1 Hardware

HMI devices MUST have the following minimum characteristics:

- **Display**: required (several displays are permitted),
  - Display size: any
  - Display type: graphic with depth is 1-bit or more.
- **Input**: optional
  - Any user-input mechanisms: buttons, rotary switches, keyboards, touch and multi-touch screens, mouse-like-pointers, etc ...
- **Memory**:
  - 7 kilobytes of non-volatile memory for the MWT implementation.

2.1.2 Software

The MWT profile specification makes minimal assumptions about the system software of the embedded HMI device. These requirements are as follows:

- A fully featured J2ME Java virtual machine. The kernel does not need to support an OS/RTOS - the virtual machine may be baremetal (i.e. the device boots directly in Java).
- A CLDC 1.1 [CLDC 1.1] library running on top of the J2ME Java virtual machine.
- An implementation of [MicroUI 1.4] classes.

2.1.3 Specification

This section sets out the requirements of this specification.

Compliant MWT 1.0 implementations:

- MUST include all packages, classes, and interfaces of the MWT API.
- MUST support the UTF-8 character encoding.
- MUST adhere to the details of the specification as contained in the remainder of this document, with particular attention to those items marked with MUST.

2.2 Scope

2.2.1 What is MWT?

MWT is a toolkit that simplifies the creation and use of graphical user interface widgets on a pixelated display.

The MWT profile specification assumes that HMI devices are limited in processing power, memory size and display features. Although this specification defines minimal requirements, devices with more resources may also benefit from MWT's special care in employing resources to their best advantage.

2.2.2 Why MWT?

There are already many existing widget toolkits that provide different APIs and concepts, such as AWT, SWT, LWUIT, etc. None of these widget toolkits are designed for constrained devices. MWT defines a minimum graphical environment that avoids portability problems so that an application running on a constrained device would run on other hardware devices such as cellphones/PDA or PC.

The aim of this library is to be sufficient to create complex applications with a minimal framework. It provides the main concepts without managing particular needs. Specific needs can be met by a MWT expert by creating new widgets, adding more complex concepts, etc. The flexibility of the MWT open framework allows the selection of only what is necessary for the application in order to guarantee lightweight applications and fast execution.

3 MWT CONCEPTS

3.1 Roles

MWT defines three distinct roles: Widget Designers, Look & Feel Designers and Application Designers.

3.1.1 Widget Designers

Widget Designers creates new widgets by specifying widget contents and behavior. They define listeners that the application can use to be notified of changes to the widget (rather than the application having to be exposed to internal events (cf. 3.10.1)), create new animations, etc.

3.1.2 Look & Feel Designers

Look & Feel Designers create consistent sets of rendering systems in order to define the way the widgets are displayed on screens. The three elements of interest to Look & Feel Designers are:

1. Renderers, that render widgets on a screen (cf. 3.6.1).
2. Looks, that define the usage of colors, fonts and other visual clues.
3. Themes, that group together a consistent set of renderers and a Look.

3.1.3 Application Designers

Application Designers create the HMI and the functional parts of the application. Their role is similar to the role for other toolkits: they use widgets defined by Widget Designers and should not consider widget look & feel.

3.2 Widgets

3.2.1 Widget Architecture

A widget is an object which is intended to be displayed on a screen and that can interact with the user. MWT splits the widget concept into the notions implied by the MVC design pattern: Model, View and Controller. A `Widget` plays the roles of the Model (or at least shares this role with the application) and the Controller, as defined by the `Renderable` interface. The View role is played by a `Renderer` (cf 3.6). A `Widget` occupies a specific region of the display and holds state. A `Renderer` is stateless and can render any widget of the appropriate type.

Widgets are arranged on a `Panel` (cf. 3.4). A widget can be part of only one `Panel` hierarchy, and can appear only once on that panel.

3.2.2 Margin and Padding

The bounds of a widget define its position (relative to its parent) and its size. These bounds are normally set automatically as part of the process of laying out a set of widgets on a panel. The

widget's renderer can only paint the area defined by its bounds – painting outside that area will be clipped.

By default widgets will be arranged so that there is no space between them, but a widget's renderer can specify a margin around the widget that is to be kept clear. If there is a margin then the widget bounds will be inset by the size of the margin within the total area that is reserved for the widget.

The renderer may also wish to reserve space within the widget's bounds to separate the widget's border from its content. This space is called padding.

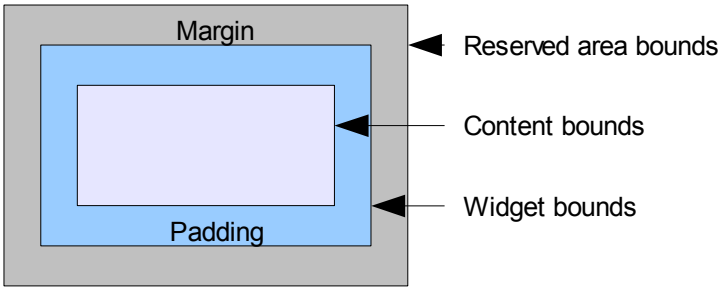


Illustration 3-1: Margin and Padding

MWT will always attempt to reserve an area for the widget that is at least as large as the content width and height requested by the widget's renderer plus twice the padding size plus twice the margin size. Note that the required margin and padding are defined by the widget's renderer and not by the widget itself.

3.2.3 Transparency

When a renderer renders a non-transparent widget it must always paint the entire area of the widget's bounds. By contrast, when a renderer renders a transparent widget it does not paint all the area of the widget's bounds but only those parts it must. Prior to the renderering of a transparent widget the widget's parent must render the area occupied by the widget. This behavior propagates to all transparent parents in the widget hierarchy. There is an exception to manage overlays (cf. 3.5.1).

A widget is transparent if it replies `true` to `Widget.isTransparent()`. A widget can be requested to be transparent using `Widget.setTransparent(true)`, but this is only a request. By default a widget will only respect such a request if it has a renderer and if that renderer does not require its widgets always to be transparent (see 3.6.2).

3.2.4 Focus

MWT defines a focus mechanism for widgets. A widget that has focus receives input events, such as button presses. It is normal for renderers to indicate focus using a visual clue.

There is one widget that owns the focus per panel, and one active panel per desktop. The focused widget of the desktop is the one in the active panel.

3.2.5 Enabled and disabled widgets

A widget can be enabled or disabled. When disabled it cannot receive focus and so cannot receive any input events. It is normal for renderers to distinguish enabled and disabled widgets using a visual clue.

3.2.6 Composed widgets

The widget designer can choose to create lightweight widgets containing several entities that avoid adding the hierarchy levels that result from using Composites (cf. 3.3). This kind of widget is called composed.

Composed widgets can define their own focus management, handling the MicroUI directional events (`Command.UP`, `Command.DOWN`, `Command.LEFT`, `Command.RIGHT`) in the `handleEvent(int)` method and defining the `requestFocus(int)` method. This latter method is called by the parent of the widget when it gives it the focus.

3.3 Composites & Layouts

A `Composite` follows the composite pattern: it is a `Widget` composed from other `Widget`s. It also defines the layout policy of its children (defining their bounds). The children's locations are relative to the location of their parent.

`Composite`s can be nested to design elaborate user interfaces. `Widget`s added to a container are stored in a list. The order of the list defines the widgets' front-to-back stacking order within the container. By default, a `Widget` is added to the end of the list (so to the bottom of the stacking order).

A composite defines the navigation order between its children, this can be done by defining `requestFocus(int)` or/and `requestFocusFrom(int, int)` methods or by handling MicroUI directional events (`Command.UP`, `Command.DOWN`, `Command.LEFT`, `Command.RIGHT`) in the `handleEvent(int)` method .

The default behavior is to focus the next widget receiving down and right directions, and the previous one receiving up and left directions. Respectively, on the last and first widgets, the event is not handled (cf. 3.10.1) and the focus management delegated to the parent.

3.4 Panels

3.4.1 Panel Definition

A `Panel` is a `Renderable` object set on a `Desktop` (cf. 3.5) and contains a `Widget` . Its bounds are relative to this `Desktop`. When a panel is repainted, its widget (and all its hierarchy for a `Composite`) are also repainted.

A panel may optionally be configured to be packed. This setting can be enabled explicitly using `Panel.setPacked(true)` or implicitly as a side-effect of `Panel.show(desktop)` or `Panel.show(desktop, false)`. An unpacked panel always has a fixed size, either because its size has been set explicitly or because it has been implicitly sized to fit the desktop. During validation (layout) the size of a panel that is packed is set to the size of its content. Conversely, for an unpacked panel the size of the content is allowed to occupy the whole of the panel.

3.4.2 Dialogs

A dialog is a modal `Panel` that blocks the thread that call the `show()` method. While it is shown, it is the only one that receive all systems events.

Several dialogs can be stacked, only the last shown receive the system events.

3.5 Desktop

A `Desktop` is a `Renderable` object and a container of `Panel`s. It is linked to one `Display` and it can be shown or hidden on a display (see Displayable in [MicroUI 1.4] specification). At any time, only one `Desktop` can be displayed per display.

When a `Desktop` is repainted, the whole set of `Panel`s it contains (and their contents) is repainted too.

3.5.1 Overlay Management

When a non-active panel (or a widget in a non-active panel) need to be repainted, all the renderables above it (respecting the stacking order) are also repainted. The resultant rendering of all the renderables must be atomic in terms of screen flush to avoid blinking issues.

3.6 Rendering

3.6.1 Renderer

A renderer is the view part of the MVC pattern of MWT.

It defines the type of objects it can display, known as its managed type. Then, only ONE instance of a `Renderer` is necessary to render ALL the widgets of the managed type. Therefore, renderers must not store widget-related fields and must not be mutable. They are stateless objects and can be allocated in non-volatile memories such as Flash memory (see `Immutables` in [B-ON 1.1])

A renderable object is intended to be rendered (by its associated renderer) on the screen. It must define the way it searches for its renderer in the `RenderingContext`. `Renderable` objects should not link directly to their renderer so that the rendering theme can be modified at runtime.

3.6.2 Widget Renderer

Renderers that are designed to render objects of type `Widget` must implement the `WidgetRenderer`. This interface requires that the renderer be able to provide the preferred width and height of the rendering of the widget's content.

3.6.3 Look

A look is a class that implements the `Look` interface. It defines a set of properties, such as fonts, colors, etc., shared by a set of renderers.

A renderer can obtain the current `Look` by calling `Renderer.getLook()`. It can then obtain specific properties from the look.

A look is related to one or more themes.

3.6.4 Theme

A theme is a coherent set of renderers and a `Look`. A theme is an instance of a class that extends the abstract class `Theme`. It should be created by the look and feel designer to define a rendering scheme.

A theme also specifies whether the renderers it contains use only the standard `Look` properties represented by the constants defined in the `Look` class. If so it must return `true` to `Theme.isStandard()`.

To conserve memory, the renderers associated with the theme are not instantiated when the theme is constructed but only when the theme is added to the rendering context. Similarly, the renderers are removed from the theme when the theme is removed from the rendering context.

The look of a theme can be changed by calling `Theme.setLook(Look)`.

3.6.5 Rendering Context

`RenderingContext` is a final class that implements the singleton pattern: there is only ever one instance of `RenderingContext`.

The rendering context holds the pool of renderers that can be used to render renderables. Renderers cannot be added to the pool directly but only by adding a theme (cf. 3.6.4).

`RenderingContext` defines a default searching algorithm is defined to get a `renderer` for a `Renderable` – see section 3.6.6 for details.

The `RenderingContext` defines a state that must change every time the renderer pool has changed. It must ensure that between two calls of the method `getState()` the result must be the same if the pool has not been modified and it should be different otherwise. This state allows the rendering context to cache the results of renderer searches using weak pointers; instead of searching for the renderer each time it is needed, the search is required only when the pool has changed.

The look of all themes connected to the rendering context that respond `true` to `Theme.isStandard()` can be changed in one operation by calling `RenderingContext.setLook(Look)`.

3.6.6 Renderer Selection

The implementations of `Renderable.getRenderer()` in `Widget`, `Panel` and `Desktop`, which will apply unless overridden, must use the default renderer selection algorithm defined by `RenderingContent.getRenderer(Renderable)`.

The selection algorithm to find the renderer for `Renderable X` is as follows:

1. Reject all renderers whose managed type is not the class of `X` or a superclass of the class of `X`.
2. From the remaining renderers find the set of renderers that have the smallest distance in the class hierarchy between the managed type of the renderer and the class of `X`.
3. If only one renderer has this smallest distance it is selected.

4. If several renderers have this smallest distance then for each such renderer `R` select the renderer that has the largest result from calling `X.computeScore(R)`.
5. If several renderers have the same score then select from these renderers the renderer that would appear first in the list of renderers obtained by calling `RenderingContext.getRenderers()`.

The implementations of `Renderable.computeScore(Renderer)` in `Widget`, `Panel` and `Desktop`, which will apply unless overridden, must use the default scoring algorithm defined by `RenderingContent.computeScore(int, int)`, passing as parameters the `getStyle()` of the renderable and the `getManagedStyle()` of the renderer. This algorithm must return the number of matching 1-bits of the two parameters.

Note that a renderer that can render type T can also be used to render objects that conform to a subtype of T.

The purpose of selecting renderers based on the style of the renderable is to allow two or more widgets of the same class to be displayed in different ways, using different renderers, according to their style.

3.6.7 Non-Rectangular Shapes Management

It is possible to define non-rectangular widgets or panels by redefining the `contains(int x, int y)` method. It must return whether or not a pixel is inside the rendererable. The results of calls to this method are used to determine the appropriate distribution of pointer events (cf. 3.9).

Note that the clipping when painting is rectangular, and nothing prevents inner widgets being outside the non-rectangular bounds.

3.7 Repainting

A widget or panel is requested to repaint by calling its `repaint()` method. All children are also repainted. The repainting is performed asynchronously; the `repaint()` method must return immediately. When a widget's state changes is a way that will not affect the layout of the panel containing the widget it is appropriate to use `repaint()` to repaint it.

3.8 Layout Validation

The layout of a panel hierarchy is performed automatically only when the panel is first shown on a desktop.

When a widget changes in a way that may affect the layout its `revalidate()` method should be called. That marks all its hierarchy (panel included) as needing to be laid out. The layout is performed asynchronously; the `revalidate()` method must return immediately.

When the asynchronous layout is performed the `validate()` method of the panel must be called. This must:

1. call the `validate(int, int)` method of its child widget (which may be a composite), passing in the available space;
2. ask the child for its preferred size (which the child will have computed during its execution of `validate(int, int)`);

3. set the bounds of the child widget, taking into account the widget's preferred size and the available space;
4. force a repaint.

The `validate(int, int)` method of a widget must:

1. call the `validate(int, int)` method of its child widgets, if any;
2. compute and store its preferred size (which may be based on the preferred sizes of its children, if it has them);
3. set the bounds of the children, if it has them, taking into account the children's preferred sizes and the available space.

3.9 Pointer

A pointer is a pointing device with buttons.

Each pointer is linked to a renderable. This renderable is either the one which is under the pointer if no button of this pointer is pressed, or the one under the pointer when the first button has been pressed.

This renderable receives pointer events via its `handleEvent(event)` method.

The detection of the renderable which is under the pointer takes into consideration the stacking order of the panels and widgets. In illustration 3-2, the renderable will be searched for in `Panel 2`, then in `Panel 1`.

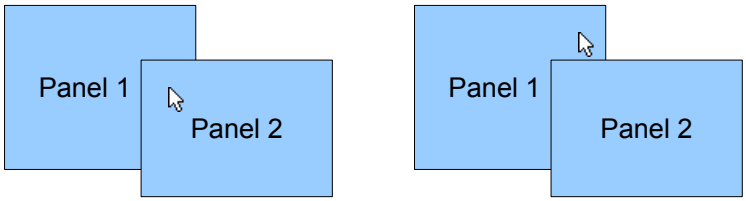


Illustration 3-2: Widget under pointer resolution

When a pointer button is pressed over a widget, the widget becomes the focus owner and its containing panel becomes the active panel of the desktop. And when a pointer is pressed over a panel, this panel becomes the active panel of the desktop.

3.10 Focus and event handling

MWT defines a focus mechanism for widgets.

There is one widget that owns the focus per panel , and one active panel per desktop. The focused widget of the desktop is the one in the active panel.

3.10.1 Events

MWT adopts the event concept from [MicroUI 1.4] , where an event is represented by a Java `int`. Please refers to the [MicroUI 1.4] documentation for more information about the MicroUI events.

Event handling is performed by giving the received event to most specific widget that has focus using the `handleEvent(int)` method. This method returns a boolean that indicates whether or not the internal event has been consumed by the widget. If that widget does not consume the event it is offered in turn to each of the widget's parents, progressing up the composition hierarchy, until it reaches the `Panel`.

MicroUI events are called internal events and MUST be used only by widgets designers: they are related to widget behavior only (cf. 3.1.1).

3.10.2 Listeners

The Application Designer MUST not be exposed to or use events that drive the internal behavior of widgets.

Therefore MWT provides a set of listeners that are intended to be used by the Widget Designer (cf. 3.1.1) to notify the Application Designer (cf. 3.1.3) that a widget's state has changed: `PanelListener`, `FocusListener`, `RenderableListener` . Note that MWT itself makes no use of these interfaces.

3.11 Automatic Testing

Automatic HMI testing is possible with MWT thanks to its reliance on a framework that permits the generation of internal events in software, simulating hardware interactions and user interaction. Refer to [MicroUI 1.4] automatic testing process for more details, in particular all `send(…)` methods of the `EventGenerator` hierarchy.

4 ARCHITECTURE

4.1 Framework

A MWT implementation MUST be thread safe in the sense that:

- Any method can be called several times from several threads concurrently on the same receiver without jeopardizing the state of that receiver.
- The user application may synchronize to any object without causing a deadlock with the implementation of MWT.

This definition allows the number and the size of critical sections to be minimized without compromising the robustness of the framework.

4.2 Drawing

MWT is strictly a widget UI toolkit and does not try to abstract the underlying system drawing capabilities. To enable portability, MWT defines a minimal environment based [MicroUI 1.4] needed to develop MWT widgets and applications.

A `Renderable` object is drawn when the `render(ej.microui.io.GraphicsContext g, Renderable renderable)` method is invoked on its renderer. The `GraphicsContext` is used to draw on the display.

4.3 System Properties

The MWT specification defines a set of properties, described in Table 4-1.

Property	Description
`ej.mwt.vendor`	*Optional*. The name of MWT library provider.
`ej.mwt.vendor.url`	*Optional*. The web site of the MWT library provider.
`ej.mwt.version`	*Optional*. The MWT version that is supported by the implementation: three numbers separated with `'.'` (an example is `1.3.1`)

Table 4-1: System Properties

5 APPENDIX

5.1 Architecture

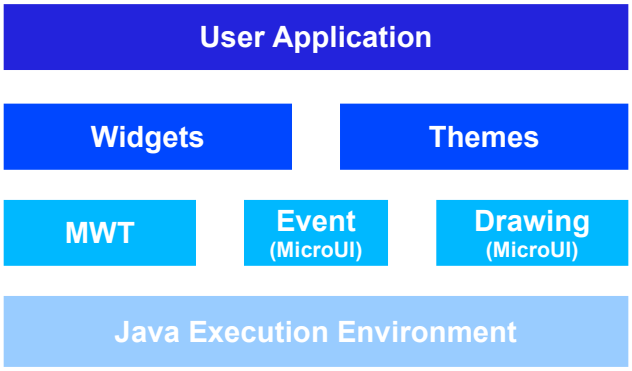


Illustration 5-1: MWT Application Architecture

5.2 Class Diagram

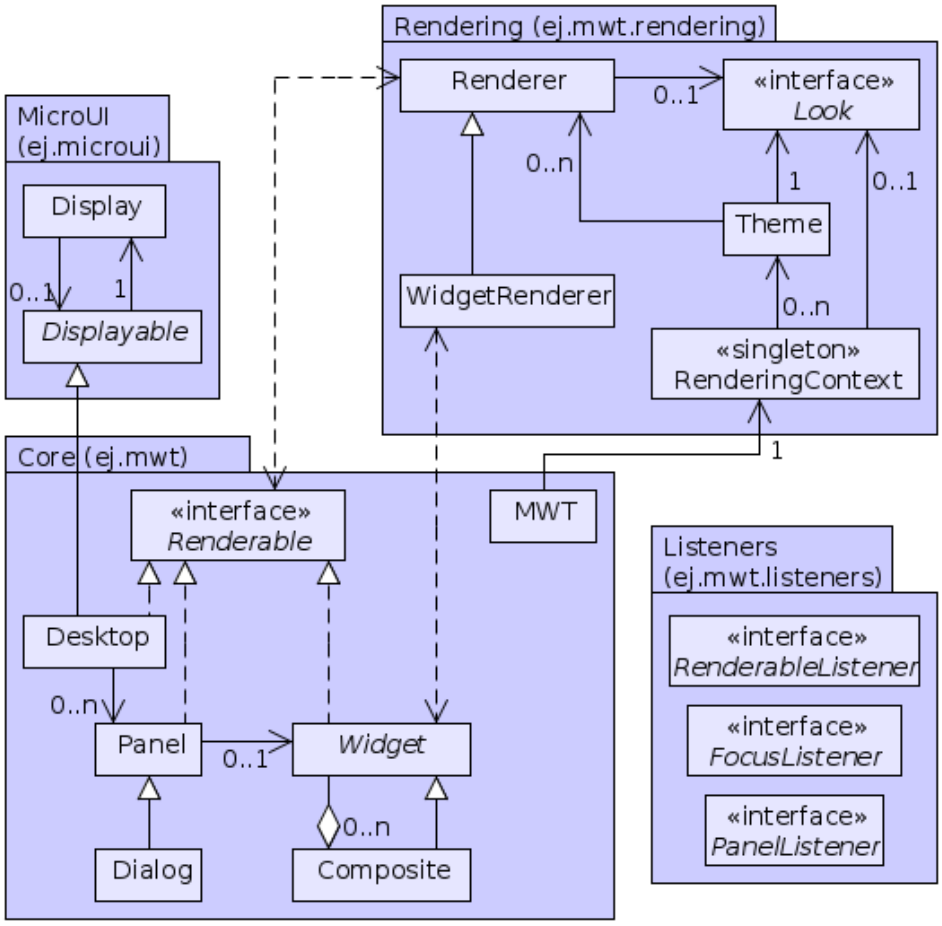


Illustration 5-2: MWT Class Diagram

6 JAVA SPECIFICATION

Package ej.mwt

Interface Summary		Page
Renderable	A renderable is an object that is intended to be rendered on the screen.	50

Class Summary		Page
Composite	A composite is a widget that can contain other Widget instances, following the composite pattern.	15
Desktop	A desktop is the top-level object that can be displayed on a `ej.microui.io.Display`.	22
Dialog	A dialog is a specific panel that once shown is the only one that receives all input events until it is hidden.	30
MWT	This class defines standard flags to use in bit-fields.	32
Panel	A panel is a Renderable object that can be shown on a Desktop.	37
Widget	Widget is the superclass of all the user interface objects.	54

Class Composite

ej.mwt

```
java.lang.Object
  └
    ej.mwt.Widget
      └
        ej.mwt.Composite
```

All Implemented Interfaces:
Renderable

```
abstract public class Composite
extends Widget
```

A composite is a widget that can contain other Widget instances, following the composite pattern.

The children are stored in a list. The order of the list defines the front-to-back stacking order of the widgets within the composite. The first widget in the list is at the back of the stacking order.

A widget cannot be added twice to a composite hierarchy.

Constructor Summary	Page
Composite() Creates a new composite.	16
Composite(int x, int y, int width, int height) **Deprecated.** *use Composite() and Widget.setBounds(int, int, int, int)*	16

Method Summary		Page
void	**add**(Widget widget) Adds the specified widget to the end of the list of children of this composite.	17
void	**addWidget**(Widget widget) **Deprecated.** *use add(Widget)*	17
Widget	**getFocus**() Gets the widget that is the focus owner or that is (recursively) the focus owner parent in the composite.	20
int	**getFocusIndex**() Gets the index of the widget that is the focus owner or that is (recursively) the focus owner parent in the composite.	20
int	**getNext**(int from, int direction) Gets the next widget in the focus order following the direction.	20
Widget	**getWidgetAt**(int x, int y) Returns the child widget that is at the specified location.	18
Widget[]	**getWidgets**() Gets the list of children in this composite.	18
int	**getWidgetsCount**() Gets the number of children in this composite.	18
boolean	**handleEvent**(int event) Called by the system if a child of this composite is the owner of the focus of the active panel (recursively) and has not consumed the specified event.	21

boolean	**hasFocus**() Gets whether or not this composite or one of its children (recursively) is the focus owner of its panel.	*18*
void	**remove**(Widget widget) Removes the specified widget from the list of children of this composite.	*17*
void	**removeAllWidgets**() Removes all the widgets from the list of children of this composite.	*17*
void	**removeWidget**(Widget widget) **Deprecated.** *use remove(Widget)*	*17*
void	**requestFocus**() Requests that the first child of this composite be set as the focus owner of its panel.	*19*
boolean	**requestFocus**(int direction) Sets a widget in this composite as the focus owner of its panel, if it is enabled, following the direction.	*19*
boolean	**requestFocusFrom**(int from, int direction) Gives the focus to the first enabled widget that is in the list of this composite's children from the widget at the specified index following the direction.	*19*
abstract void	**validate**(int widthHint, int heightHint) Lays out this composite and all its children.	*21*

Methods inherited from class ej.mwt.Widget
cleanRendererCache, computeScore, contains, gainFocus, getAbsoluteX, getAbsoluteX, getAbsoluteY, getAbsoluteY, getHeight, getPanel, getParent, getPreferredHeight, getPreferredWidth, getRelativeX, getRelativeY, getRenderer, getStyle, getWidth, getX, getY, isEnable, isEnabled, isShown, isTransparent, isValid, isVisible, lostFocus, repaint, repaint, revalidate, setBounds, setEnable, setEnabled, setLocation, setPreferredSize, setSize, setTransparent, setVisible

Constructor Detail

Composite

```
public Composite()
```

Creates a new composite.

Its bounds will be set to 0.

Composite

```
public Composite(int x,
                 int y,
                 int width,
                 int height)
```

Deprecated.

Creates a new composite specifying its bounds. Its position is relative to the position of its parent.

Parameters:

x - the relative x coordinate of the composite
y - the relative y coordinate of the composite
width - the width of the composite
height - the height of the composite

Method Detail

add

`public void` **`add`**`(Widget widget)`

Adds the specified widget to the end of the list of children of this composite.

If the composite is on a panel hierarchy, it is asked to be revalidated.

Parameters:
`widget` - the widget to add
Throws:
`NullPointerException` - if the specified widget is null
`IllegalArgumentException` - if the specified widget or one of its children is already connected to a Panel
See Also:
Widget.revalidate()

---

addWidget

`public void` **`addWidget`**`(Widget widget)`

Deprecated. *use add(Widget)*

---

remove

`public void` **`remove`**`(Widget widget)`

Removes the specified widget from the list of children of this composite.

If the composite is on a panel hierarchy, it is asked to be revalidated.

If the widget is not in the list of children of the composite, nothing is done.

Parameters:
`widget` - the widget to remove
Throws:
`NullPointerException` - if the specified widget is null
See Also:
Widget.revalidate()

---

removeWidget

`public void` **`removeWidget`**`(Widget widget)`

Deprecated. *use remove(Widget)*

---

removeAllWidgets

`public void` **`removeAllWidgets`**`()`

Removes all the widgets from the list of children of this composite.

If the composite is on a panel hierarchy, it is asked to be revalidated.

See Also:
Widget.revalidate()

---

getWidgetAt

```
public Widget getWidgetAt(int x,
                          int y)
```

Returns the child widget that is at the specified location.

If Widget.contains(int, int) is `false` for this composite, `null` is returned. Otherwise, if there is a child for which Widget.contains(int, int) returns `true` then the result of invoking Widget.getWidgetAt(int, int) on that widget is returned. Otherwise this composite is returned.

The location is relative to the location of this composite's parent.

Overrides:
getWidgetAt in class Widget
Parameters:
`x` - x coordinate
`y` - y coordinate
Returns:
the widget at the location, `null` if no widget is found in this composite hierarchy.

---

getWidgets

```
public Widget[] getWidgets()
```

Gets the list of children in this composite.

Returns:
the list of children

---

getWidgetsCount

```
public int getWidgetsCount()
```

Gets the number of children in this composite.

Returns:
the number of children

---

hasFocus

```
public boolean hasFocus()
```

Gets whether or not this composite or one of its children (recursively) is the focus owner of its panel.

Returns `false` if this composite is not on a panel.

Overrides:
hasFocus in class Widget
Returns:
`true` if this composite is a parent of the focus owner, `false` otherwise

---

requestFocus

```
public void requestFocus()
```

Requests that the first child of this composite be set as the focus owner of its panel.

If the composite does not contains any widgets, nothing is done.
If the composite is not in a panel hierarchy, nothing is done.

Identical to call requestFocusFrom(int, int) with MWT.RIGHT as direction and `0` as from.

Overrides:
requestFocus in class Widget

---

requestFocus

```
public boolean requestFocus(int direction)
```

Sets a widget in this composite as the focus owner of its panel, if it is enabled, following the direction.

The given direction must be one of MWT.UP, MWT.DOWN, MWT.LEFT, MWT.RIGHT.
If the widget is not in a panel hierarchy, nothing is done.

Identical to call requestFocusFrom(int, int) with `0` as `from` and one of MWT.DOWN or MWT.RIGHT as `direction`, or `(getWidgetsCount() - 1)` as `from` and one of MWT.LEFT or MWT.UP as `direction`.

Overrides:
requestFocus in class Widget
Parameters:
`direction` - the direction followed by the focus
Returns:
`true` if the composite take the focus, `false` otherwise
Throws:
`IllegalArgumentException` - if `direction` is not a valid direction

---

requestFocusFrom

```
public boolean requestFocusFrom(int from,
                                int direction)
```

Gives the focus to the first enabled widget that is in the list of this composite's children from the widget at the specified index following the direction.

The given direction must be one of MWT.UP, MWT.DOWN, MWT.LEFT, MWT.RIGHT.

Parameters:
`from` - the index to start search
`direction` - the direction followed by the focus
Returns:
`true` if a new widget has been given focus, `false` otherwise

Throws:
`ArrayIndexOutOfBoundsException` - if `from` is not a valid index
`IllegalArgumentException` - if `direction` is not a valid direction
See Also:
getNext(int, int)

---

getNext

```
public int getNext(int from,
                   int direction)
```

Gets the next widget in the focus order following the direction.

If there is no widget to take focus in this direction, it returns MWT.EMPTY.
The given direction must be one of MWT.UP, MWT.DOWN, MWT.LEFT, MWT.RIGHT.

Parameters:
`from` - the index of the current widget
`direction` - the direction to follow
Returns:
the index of the next widget
Throws:
`ArrayIndexOutOfBoundsException` - if `from` is not a valid index
`IllegalArgumentException` - if `direction` is not a valid direction
Since:
1.0
See Also:
getFocusIndex()

---

getFocus

```
public Widget getFocus()
```

Gets the widget that is the focus owner or that is (recursively) the focus owner parent in the composite.

Returns `null` if the focus owner is not in the composite hierarchy.

Returns:
the widget that own the focus on this composite or `null`

---

getFocusIndex

```
public int getFocusIndex()
```

Gets the index of the widget that is the focus owner or that is (recursively) the focus owner parent in the composite.

Returns MWT.EMPTY if the focus owner is not in the composite hierarchy.

Returns:
the index of the widget that owns the focus on this composite or MWT.EMPTY

---

validate

```
public abstract void validate(int widthHint,
                              int heightHint)
```

Lays out this composite and all its children.

The parameters defines the maximum size available for this composite, or MWT.NONE if there is no constraint. After this call the preferred size will have been established.

Overrides:
validate in class Widget
Parameters:
`widthHint` - the width available for this widget or MWT.NONE
`heightHint` - the height available for this widget or MWT.NONE

---

handleEvent

```
public boolean handleEvent(int event)
```

Called by the system if a child of this composite is the owner of the focus of the active panel (recursively) and has not consumed the specified event.

Composites handle `ej.microui.Command.UP`, `ej.microui.Command.DOWN`, `ej.microui.Command.LEFT`, and `ej.microui.Command.RIGHT` commands to manage navigation between its children.

Specified by:
handleEvent in interface Renderable
Overrides:
handleEvent in class Widget
Parameters:
`event` - the event to handle
Returns:
`true` if the composite consumed the event, `false` otherwise

Class Desktop

ej.mwt

```
java.lang.Object
  └
    ej.microui.io.Displayable
      └
        ej.mwt.Desktop
```

All Implemented Interfaces:

Renderable

```
public class Desktop
extends ej.microui.io.Displayable
implements Renderable
```

A desktop is the top-level object that can be displayed on a `ej.microui.io.Display`.
A desktop is built for a specific `ej.microui.io.Display`, and that relationship cannot be modified.
A desktop may be shown or hidden, but at most one desktop is shown per `ej.microui.io.Display`.

A desktop can contains several Panel instances.
These panels are stored in a list. The order of the list defines the front-to-back stacking order of the panels within the desktop.
The first panel in the list is at the back of the stacking order.

See Also:

`ej.microui.io.Display`, Panel

Constructor Summary	Page
Desktop() Creates a new desktop on the default display.	23
Desktop(`ej.microui.io.Display display`) Creates a new desktop on the specified display.	23

Method Summary		Page
`protected void`	**cleanRendererCache**() Cleans the renderer cache.	26
`int`	**computeScore**(Renderer `renderer`) Computes the score of the given renderer by comparing the getStyle() of the desktop with the Renderer.getManagedStyle() of the specified renderer.	26
Panel	**getActivePanel**() Gets this desktop's active panel.	27
`int`	**getHeight**() Returns the height of this desktop: it is equal to the height of its associated display.	24
Panel[]	**getPanels**() Gets the list of the panels associated with this desktop.	27
Renderer	**getRenderer**() Gets the renderer associated with this desktop in the rendering context.	25
`int`	**getStyle**() Gets the style of the desktop.	26

int	**getWidth**() Returns the width of this desktop: it is equal to the width of its associated display.	*24*
int	**getX**() Returns the x coordinate of this desktop, which is always 0.	*24*
int	**getY**() Returns the y coordinate of this desktop, which is always 0.	*24*
boolean	**handleEvent**(int event) Called by the system if no widget nor panel in the focused hierarchy has consumed the event.	*28*
boolean	**isShown**() Gets whether or not the desktop is shown on a display.	*25*
void	**paint**(ej.microui.io.GraphicsContext g) Paint the desktop and all its children.	*29*
void	**performAction**(int event) Sends the given event to the widget that owns the focus: • for ej.microui.io.Pointer events, the one which is under the pointer; • for other events, the one which is the focus owner of the active panel.	*28*
void	**repaint**() Requests a repaint of this entire desktop.	*25*
void	**repaint**(int x, int y, int width, int height) Requests a repaint of a zone of this desktop.	*25*
void	**revalidate**() Lays out all the hierarchy of this desktop.	*27*
void	**setActivePanel**(Panel panel) Sets the specified panel as the active one on this desktop.	*27*
protected void	**showNotify**() Called by system as soon as the desktop becomes visible on its display.	*28*
void	**validate**() Lays out all the hierarchy of this desktop.	*27*

Methods inherited from class ej.microui.io.Displayable
getDisplay, hide, hideNotify, show

Constructor Detail

Desktop

```
public Desktop()
```

Creates a new desktop on the default display.
Identical to `new Desktop(ej.microui.io.Display.getDefaultDisplay())`.

Desktop

```
public Desktop(ej.microui.io.Display display)
```

Creates a new desktop on the specified display. The newly created desktop is hidden.

Parameters:
`display` - the display for which the desktop is created
Throws:
`NullPointerException` - if `display` is `null`

Method Detail

getX

```
public int getX()
```

Returns the x coordinate of this desktop, which is always 0.

Specified by:
getX in interface Renderable
Returns:
the x coordinate of this desktop

---

getY

```
public int getY()
```

Returns the y coordinate of this desktop, which is always 0.

Specified by:
getY in interface Renderable
Returns:
the y coordinate of this desktop

---

getWidth

```
public int getWidth()
```

Returns the width of this desktop: it is equal to the width of its associated display.

Specified by:
getWidth in interface Renderable
Returns:
the width of this desktop

---

getHeight

```
public int getHeight()
```

Returns the height of this desktop: it is equal to the height of its associated display.

Specified by:
getHeight in interface Renderable
Returns:
the height of this desktop

---

isShown

```
public boolean isShown()
```

Gets whether or not the desktop is shown on a display.

Specified by:
isShown in interface Renderable
Overrides:
`isShown` in class `ej.microui.io.Displayable`
Returns:
`true` if the desktop is shown, `false` otherwise.
See Also:
`ej.microui.io.Displayable.getDisplay()`

---

repaint

```
public void repaint()
```

Requests a repaint of this entire desktop.
This method returns immediately; the repaint of the desktop is performed asynchronously.
If the desktop is not shown, nothing is done.

Specified by:
repaint in interface Renderable
Overrides:
`repaint` in class `ej.microui.io.Displayable`

---

repaint

```
public void repaint(int x,
                    int y,
                    int width,
                    int height)
```

Requests a repaint of a zone of this desktop.
This method returns immediately; the repaint of the desktop is performed asynchronously.
If the desktop is not shown, nothing is done.

Specified by:
repaint in interface Renderable
Parameters:
`x` - the relative x coordinate of the area to repaint
`y` - the relative y coordinate of the area to repaint
`width` - the width of the area to repaint
`height` - the height of the area to repaint

---

getRenderer

```
public Renderer getRenderer()
```

Gets the renderer associated with this desktop in the rendering context.

The renderer is located using the RenderingContext default algorithm.
The renderer is kept in cache until the state of the rendering context is changed.

Specified by:
getRenderer in interface Renderable
Returns:
the renderer associated with this desktop, or `null` if none
See Also:
RenderingContext.getRenderer(Renderable), RenderingContext.getState()

---

cleanRendererCache

```
protected void cleanRendererCache()
```

Cleans the renderer cache.

Subclasses can call this method when the style of the desktop is changed and the best-fit renderer may be different (even though the pool of renderers may not have changed).
The next time getRenderer() is called a new search will be performed.

---

getStyle

```
public int getStyle()
```

Gets the style of the desktop. The style is used to get the best match renderer to associate with this desktop.

Always returns 0 but may be overridden in subclasses.

Specified by:
getStyle in interface Renderable
Returns:
the style
Since:
1.0
See Also:
Renderer.getManagedStyle(), getRenderer()

---

computeScore

```
public int computeScore(Renderer renderer)
```

Computes the score of the given renderer by comparing the getStyle() of the desktop with the Renderer.getManagedStyle() of the specified renderer.
The score is bigger when the renderer matches the style and lower when it does not match.
The score is computed using the RenderingContext.computeScore(int, int) algorithm.

Specified by:
computeScore in interface Renderable
Parameters:
`renderer` - the renderer to compute score with
Returns:
the score between the given style and the receiver style
Since:
1.0
See Also:
RenderingContext.computeScore(int, int)

---

getActivePanel

`public` Panel **`getActivePanel`**`()`

Gets this desktop's active panel. The active panel is the last in the panel's list.

Returns:
the desktop's active panel, or `null` if none

---

setActivePanel

`public void` **`setActivePanel`**`(`Panel `panel)`

Sets the specified panel as the active one on this desktop.

Parameters:
`panel` - the panel to set active
Throws:
`NullPointerException` - if the specified panel is null
`IllegalArgumentException` - if the specified panel is not on this desktop
See Also:
getActivePanel()

---

getPanels

`public` Panel`[]` **`getPanels`**`()`

Gets the list of the panels associated with this desktop.
A panel is added to this list when it executes Panel.show(Desktop) and removed when it executes Panel.hide().

Returns:
the list of the panels on this desktop

---

revalidate

`public void` **`revalidate`**`()`

Lays out all the hierarchy of this desktop.

It performs the method validate() asynchronously. Therefore this method is not blocked waiting until the validation of the hierarchy is done.

Nothing is done if it is not shown.

Since:
1.0
See Also:
validate()

---

validate

`public void` **`validate`**`()`

Lays out all the hierarchy of this desktop.

Since:
1.0
See Also:
Panel.validate()

---

showNotify

```
protected void showNotify()
```

Called by system as soon as the desktop becomes visible on its display.

Asks for a revalidation of the entire desktop.

Overrides:
`showNotify` in class `ej.microui.io.Displayable`
Since:
1.0
See Also:
revalidate()

---

handleEvent

```
public boolean handleEvent(int event)
```

Called by the system if no widget nor panel in the focused hierarchy has consumed the event.

Specified by:
handleEvent in interface Renderable
Parameters:
`event` - the event to handle
Returns:
`true` if the desktop has consumed the event, `false` otherwise
Since:
0.9

---

performAction

```
public void performAction(int event)
```

Sends the given event to the widget that owns the focus:

- for `ej.microui.io.Pointer` events, the one which is under the pointer;
- for other events, the one which is the focus owner of the active panel.

Overrides:
`performAction` in class `ej.microui.io.Displayable`
Parameters:
`event` - the event to handle

---

paint

```
public void paint(ej.microui.io.GraphicsContext g)
```

Paint the desktop and all its children.

If there is no `Panel` on the desktop or if the desktop is not visible, this call has no effect.

Overrides:

`paint` in class `ej.microui.io.Displayable`

Class Dialog

ej.mwt

```
java.lang.Object
  └
    ej.mwt.Panel
      └
        ej.mwt.Dialog
```

All Implemented Interfaces:

Renderable

```
public class Dialog
extends Panel
```

A dialog is a specific panel that once shown is the only one that receives all input events until it is hidden.

If several dialogs are shown, they are stacked and only the last one is active.

Constructor Summary	Page
Dialog() Creates a new dialog.	30
Dialog(int x, int y, int width, int height) Creates a new dialog specifying its bounds (relative to the desktop).	30

Method Summary		Page
void	**show**(Desktop desktop) Requests the panel to be shown on the specified desktop.	31

Methods inherited from class ej.mwt.Panel
becameActive, becameInactive, cleanRendererCache, computeScore, contains, getDesktop, getFocus, getHeight, getPreferredHeight, getPreferredWidth, getRenderer, getStyle, getWidget, getWidgetAt, getWidth, getX, getY, handleEvent, hide, invalidate, isActive, isPacked, isShown, isTransparent, isValid, repaint, repaint, revalidate, setBounds, setFocus, setLocation, setPacked, setPreferredSize, setSize, setTransparent, setWidget, show, validate, validate

Constructor Detail

Dialog

```
public Dialog()
```

Creates a new dialog.
Identical to `new Dialog(0, 0, 0, 0)`.

Dialog

```
public Dialog(int x,
              int y,
              int width,
              int height)
```

Creates a new dialog specifying its bounds (relative to the desktop).

Parameters:

`x` - the relative x coordinate of the dialog
`y` - the relative y coordinate of the dialog
`width` - the width of the dialog
`height` - the height of the dialog

Method Detail

show

```
public void show(Desktop desktop)
```

Requests the panel to be shown on the specified desktop.

The method does not return while the dialog is shown.
During this period, all the input events are sent only to this dialog.

Overrides:

show in class Panel

Parameters:

`desktop` - the desktop

Throws:

`NullPointerException` - if `desktop` is null.

See Also:

Panel.show(Desktop)

Class MWT

ej.mwt

```
java.lang.Object
  └
    ej.mwt.MWT
```

```
public class MWT
extends Object
```

This class defines standard flags to use in bit-fields.

Field Summary		Page
static int	**BOTTOM** Constant for alignment behavior.	35
static int	**CENTER** Constant for alignment or direction behavior.	34
static int	**DOWN** Constant for direction behavior.	34
static int	**EAST** Constant for alignment or direction behavior.	34
static int	**EMPTY** Constant equals to -1.	33
static int	**ERROR** Constant equals to -1.	33
static int	**HCENTER** Constant for alignment behavior.	35
static int	**HORIZONTAL** Constant for alignment or orientation behavior.	35
static int	**HSCROLL** Constant for scrolling behavior.	36
static int	**LEAD** Constant for direction behavior.	34
static int	**LEFT** Constant for alignment behavior.	35
static int	**MULTI** Constant for multiplicity.	36
static int	**NONE** Constant equals to zero used to put and check that no flags are set.	33
static int	**NORTH** Constant for alignment or direction behavior.	33
static RenderingContext	**RenderingContext** The rendering context singleton.	36
static int	**RIGHT** Constant for alignment behavior.	35
static int	**SINGLE** Constant for uniqueness.	36
static int	**SOUTH** Constant for alignment or direction behavior.	34

static int	**TOP** Constant for alignment behavior.	*35*
static int	**TRAIL** Constant for direction behavior.	*34*
static int	**UP** Constant for direction behavior.	*34*
static int	**VCENTER** Constant for alignment behavior.	*35*
static int	**VERTICAL** Constant for alignment or orientation behavior.	*35*
static int	**VSCROLL** Constant for scrolling behavior.	*36*
static int	**WEST** Constant for alignment or direction behavior.	*34*
static int	**WRAP** Constant for wrapping behavior.	*36*

Constructor Summary	*Page*
MWT()	*36*

Field Detail

ERROR

```
public static final int ERROR
```

Constant equals to -1.

EMPTY

```
public static final int EMPTY
```

Constant equals to -1.

NONE

```
public static final int NONE
```

Constant equals to zero used to put and check that no flags are set.

NORTH

```
public static final int NORTH
```

Constant for alignment or direction behavior.

SOUTH

```
public static final int SOUTH
```

Constant for alignment or direction behavior.

WEST

```
public static final int WEST
```

Constant for alignment or direction behavior.

EAST

```
public static final int EAST
```

Constant for alignment or direction behavior.

CENTER

```
public static final int CENTER
```

Constant for alignment or direction behavior.

UP

```
public static final int UP
```

Constant for direction behavior. (Same as NORTH)

DOWN

```
public static final int DOWN
```

Constant for direction behavior. (Same as SOUTH)

LEAD

```
public static final int LEAD
```

Constant for direction behavior. (Same as WEST)

TRAIL

```
public static final int TRAIL
```

Constant for direction behavior. (Same as EAST)

TOP

```
public static final int TOP
```

Constant for alignment behavior. (Same as NORTH)

BOTTOM

```
public static final int BOTTOM
```

Constant for alignment behavior. (Same as SOUTH)

LEFT

```
public static final int LEFT
```

Constant for alignment behavior. (Same as WEST)

RIGHT

```
public static final int RIGHT
```

Constant for alignment behavior. (Same as EAST)

HCENTER

```
public static final int HCENTER
```

Constant for alignment behavior.

VCENTER

```
public static final int VCENTER
```

Constant for alignment behavior.

HORIZONTAL

```
public static final int HORIZONTAL
```

Constant for alignment or orientation behavior.

VERTICAL

```
public static final int VERTICAL
```

Constant for alignment or orientation behavior.

HSCROLL

```
public static final int HSCROLL
```

Constant for scrolling behavior.

VSCROLL

```
public static final int VSCROLL
```

Constant for scrolling behavior.

SINGLE

```
public static final int SINGLE
```

Constant for uniqueness.

MULTI

```
public static final int MULTI
```

Constant for multiplicity.

WRAP

```
public static final int WRAP
```

Constant for wrapping behavior.

RenderingContext

```
public static final RenderingContext RenderingContext
```

The rendering context singleton.

Constructor Detail

MWT

```
public MWT()
```

Class Panel

ej.mwt

```
java.lang.Object
  └
    ej.mwt.Panel
```

All Implemented Interfaces:

Renderable

Direct Known Subclasses:

Dialog

```
public class Panel
extends Object
implements Renderable
```

A panel is a Renderable object that can be shown on a Desktop. It can contain a Widget.

Constructor Summary	Page
Panel() Creates a new panel.	39
Panel(int x, int y, int width, int height) Creates a new panel specifying its bounds (relative to the desktop).	39

Method Summary		Page
void	**becameActive**() Notifies the panel that it is now the active panel of its desktop.	43
void	**becameInactive**() Notifies the panel that it is no longer the active panel of its desktop.	44
protected void	**cleanRendererCache**() Cleans the renderer cache.	46
int	**computeScore**(Renderer renderer) Computes the score of the given renderer by comparing the getStyle() of the panel with the Renderer.getManagedStyle() of the specified renderer.	46
boolean	**contains**(int x, int y) Returns true if the (x,y) position is in the panel's bounds, false otherwise.	45
Desktop	**getDesktop**() Gets the desktop on which the panel is shown.	44
Widget	**getFocus**() Gets the widget that is the focus owner of this panel.	47
int	**getHeight**() Returns the height of the panel.	41
int	**getPreferredHeight**() Returns the preferred height of the panel.	42
int	**getPreferredWidth**() Returns the preferred width of the panel.	41
Renderer	**getRenderer**() Gets the renderer associated with this panel in the rendering context.	46

int	**getStyle**() Gets the style of this panel.	46
Widget	**getWidget**() Gets the widget attached to this panel.	40
Widget	**getWidgetAt**(int x, int y) Returns the child widget at the specified location.	45
int	**getWidth**() Returns the width of the panel.	40
int	**getX**() Returns the x coordinate of the panel relative to its desktop.	40
int	**getY**() Returns the y coordinate of the panel relative to its desktop.	40
boolean	**handleEvent**(int event) Called by the system if this is the active panel and if no widget in the hierarchy of the focused widget has consumed the event.	49
void	**hide**() Requests the panel be hidden.	43
void	**invalidate**() Declares that this panel needs to be laid out.	48
boolean	**isActive**() Gets whether or not the panel is the active one on its desktop.	43
boolean	**isPacked**() Gets whether this panel is packed or not.	47
boolean	**isShown**() Gets whether or not the panel is shown on a shown desktop.	43
boolean	**isTransparent**() Gets whether this panel is transparent or not.	45
boolean	**isValid**() Gets whether this panel is valid.	48
void	**repaint**() Requests a repaint of this entire panel.	44
void	**repaint**(int x, int y, int width, int height) Requests a repaint of a zone of this panel.	44
void	**revalidate**() Lays out all the hierarchy of this panel.	48
void	**setBounds**(int x, int y, int width, int height) Set the bounds of this panel.	41
void	**setFocus**(Widget widget) Sets the specified widget as the current focus owner of this panel.	47
void	**setLocation**(int x, int y) Set the location of this panel.	41
void	**setPacked**(boolean packed) Sets this panel as packed or not.	47
void	**setPreferredSize**(int preferredWidth, int preferredHeight) Sets the preferred size of the panel.	42
void	**setSize**(int width, int height) Set the size of this panel.	41
void	**setTransparent**(boolean transparent) **Deprecated.** *see isTransparent()*	45

void	**setWidget**(Widget widget) Attach the specified widget to this panel.	39
void	**show**(Desktop desktop) Requests the panel to be shown on the specified desktop.	42
void	**show**(Desktop desktop, boolean fill) Requests the panel to be shown on the specified desktop.	42
void	**validate**() Lays out all the hierarchy of this panel.	48
void	**validate**(int widthHint, int heightHint) Lays out all the hierarchy of this panel.	49

Constructor Detail

Panel

```
public Panel()
```

Creates a new panel.

By default:

- its bounds are 0,
- it is packed.

See Also:
setPacked(boolean)

Panel

```
public Panel(int x,
             int y,
             int width,
             int height)
```

Creates a new panel specifying its bounds (relative to the desktop).

The panel is set as not packed.

Parameters:
x - the relative x coordinate of the panel
y - the relative y coordinate of the panel
width - the width of the panel
height - the height of the panel

See Also:
setPacked(boolean)

Method Detail

setWidget

```
public void setWidget(Widget widget)
```

Attach the specified widget to this panel.

If there is already a widget on this panel, the former is detached from the latter.
If the specified widget is `null`, the panel does not hold a widget anymore.

The panel is also ask to be revalidated if shown.

Parameters:
`widget` - the widget to set
See Also:
revalidate()

---

getWidget

```
public Widget getWidget()
```

Gets the widget attached to this panel.

Returns:
the widget attached with this panel or `null`

---

getX

```
public int getX()
```

Returns the x coordinate of the panel relative to its desktop.

Specified by:
getX in interface Renderable
Returns:
the relative x coordinate of the panel

---

getY

```
public int getY()
```

Returns the y coordinate of the panel relative to its desktop.

Specified by:
getY in interface Renderable
Returns:
the relative y coordinate of the panel

---

getWidth

```
public int getWidth()
```

Returns the width of the panel.

Specified by:
getWidth in interface Renderable
Returns:
the width of the panel

---

getHeight

```
public int getHeight()
```

Returns the height of the panel.

Specified by:
getHeight in interface Renderable

Returns:
the height of the panel

---

setLocation

```
public void setLocation(int x,
                        int y)
```

Set the location of this panel.

Parameters:
`x` - the x coordinate to set
`y` - the y coordinate to set

---

setSize

```
public void setSize(int width,
                    int height)
```

Set the size of this panel.

Parameters:
`width` - the width to set
`height` - the height to set

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Set the bounds of this panel.

Parameters:
`x` - the x coordinate to set
`y` - the y coordinate to set
`width` - the width to set
`height` - the height to set

---

getPreferredWidth

```
public int getPreferredWidth()
```

Returns the preferred width of the panel. The result returned is meaningful only if isValid() is `true` or if setPreferredSize(int, int) has been called explicitly.

Returns:
the preferred width of the panel

---

getPreferredHeight

```
public int getPreferredHeight()
```

Returns the preferred height of the panel. The result returned is meaningful only if isValid() is `true` or if setPreferredSize(int, int) has been called explicitly.

Returns:
the preferred height of the panel

---

setPreferredSize

```
public void setPreferredSize(int preferredWidth,
                             int preferredHeight)
```

Sets the preferred size of the panel.

Parameters:
`preferredWidth` - the width to set
`preferredHeight` - the height to set

---

show

```
public void show(Desktop desktop)
```

Requests the panel to be shown on the specified desktop.

Identical to calling show(Desktop, boolean) with `fill` set to `false`.

Parameters:
`desktop` - the desktop
Throws:
`NullPointerException` - if `desktop` is null.
See Also:
show(Desktop, boolean), hide(), isShown(), revalidate()

---

show

```
public void show(Desktop desktop,
                 boolean fill)
```

Requests the panel to be shown on the specified desktop.

If `fill` is `true`, it is fitted to desktop size minus the panel margin and it will be set as not packed.
If `fill` is `false` and no size has been set, it will be set as packed.
If the desktop is shown, the panel is automatically validated.
The panel is added to the list of panels known by the desktop.

Special cases:

- If the panel is already shown on this desktop, nothing is changed.
- If the panel is already shown on another desktop, it is hidden on this desktop before being shown on the new one.

Parameters:
`desktop` - the desktop
`fill` - `true` to fit the panel to the desktop size, `false` otherwise
Throws:
`NullPointerException` - if `desktop` is null.
See Also:
hide(), isShown(), revalidate(), setPacked(boolean)

---

hide

```
public void hide()
```

Requests the panel be hidden.
The panel is invalidated.
The panel is removed from the list of panels known by the desktop.

See Also:
show(Desktop), isShown()

---

isShown

```
public boolean isShown()
```

Gets whether or not the panel is shown on a shown desktop.

Specified by:
isShown in interface Renderable
Returns:
`true` if the panel is shown, `false` otherwise.
See Also:
getDesktop(), Desktop.isShown()

---

isActive

```
public boolean isActive()
```

Gets whether or not the panel is the active one on its desktop. If the panel is not shown on a desktop, return `false`.

Returns:
`true` if the panel is the active one, `false` otherwise.
See Also:
isShown(), getDesktop()

---

becameActive

```
public void becameActive()
```

Notifies the panel that it is now the active panel of its desktop. The subclasses can override this method to add behavior.

---

becameInactive

```
public void becameInactive()
```

Notifies the panel that it is no longer the active panel of its desktop. The subclasses can override this method to add behavior.

---

getDesktop

```
public Desktop getDesktop()
```

Gets the desktop on which the panel is shown. Returns `null` if the panel is not shown.

Returns:
the desktop on which the panel is shown or `null`.
See Also:
isShown()

---

repaint

```
public void repaint()
```

Requests a repaint of this entire panel. This method returns immediately; repainting of the panel is performed asynchronously.
If the panel is not shown, nothing is done.
If the panel is transparent, it requests a repaint of its parent within the panel's bounds.

Specified by:
repaint in interface Renderable

---

repaint

```
public void repaint(int x,
                    int y,
                    int width,
                    int height)
```

Requests a repaint of a zone of this panel. This method returns immediately; repainting of the panel is performed asynchronously.
If the panel is not shown, nothing is done.
If the panel is transparent, it requests a repaint of its parent within the requested bounds.

Specified by:
repaint in interface Renderable
Parameters:
`x` - the relative x coordinate of the area to repaint
`y` - the relative y coordinate of the area to repaint
`width` - the width of the area to repaint
`height` - the height of the area to repaint

---

setTransparent

```
public void setTransparent(boolean transparent)
```

Deprecated. *see isTransparent()*

---

isTransparent

```
public boolean isTransparent()
```

Gets whether this panel is transparent or not.
A panel is transparent if if its renderer is `null`.

A transparent panel means that it will not repaint ALL the rectangular zone defined by its bounds. Then each time it needs to be repainted, its parent (recursively if also transparent) will be repainted within the bounds of the panel. Each time a non-transparent panel needs to be repainted, it is the only one to be repainted.

Returns:
`true` if this panel is transparent, `false` otherwise.
See Also:
contains(int, int)

---

contains

```
public boolean contains(int x,
                        int y)
```

Returns `true` if the (x,y) position is in the panel's bounds, `false` otherwise.
The location is considered here as a relative location to the desktop.

Parameters:
`x` - x coordinate
`y` - y coordinate
Returns:
`true` if the (x,y) position is in the panel bounds, `false` otherwise.
See Also:
isTransparent()

---

getWidgetAt

```
public Widget getWidgetAt(int x,
                          int y)
```

Returns the child widget at the specified location. If this panel does not `contains(x, y)`, `null` is returned. The location is considered here as a relative location to the desktop.

Parameters:
`x` - x coordinate
`y` - y coordinate
Returns:
the widget at the location, `null` if no widget is found in this panel hierarchy.
See Also:
Widget.getWidgetAt(int, int)

---

getRenderer

```
public Renderer getRenderer()
```

Gets the renderer associated with this panel in the rendering context.

The renderer is located using the RenderingContext default algorithm.
The renderer is kept in cache until the state of the rendering context is changed.

Specified by:
getRenderer in interface Renderable
Returns:
the renderer associated with this panel, or `null` if none
See Also:
RenderingContext.getRenderer(Renderable)

---

cleanRendererCache

```
protected void cleanRendererCache()
```

Cleans the renderer cache.

Subclasses can call this method when the style of the desktop is changed and the best-fit renderer may be different (even though the pool of renderers may not have changed).
The next time getRenderer() is called a new search will be performed.

---

getStyle

```
public int getStyle()
```

Gets the style of this panel. The style is used to get the best match renderer to associate with this panel.

Always returns 0 but may be overridden in subclasses.

Specified by:
getStyle in interface Renderable
Returns:
the style
Since:
1.0
See Also:
Renderer.getManagedStyle(), getRenderer()

---

computeScore

```
public int computeScore(Renderer renderer)
```

Computes the score of the given renderer by comparing the getStyle() of the panel with the Renderer.getManagedStyle() of the specified renderer.
The score is bigger when the renderer matches the style and lower when it does not match.
The score is computed using the RenderingContext.computeScore(int, int) algorithm.

Specified by:
computeScore in interface Renderable

Parameters:
`renderer` - the renderer to compute score with
Returns:
the score between the given style and the receiver style
Since:
1.0
See Also:
RenderingContext.computeScore(int, int)

---

setFocus

```
public void setFocus(Widget widget)
```

Sets the specified widget as the current focus owner of this panel.
If the widget is not enabled, nothing is done.
If the widget already own the focus, nothing is done.
Throws an IllegalArgumentException if the specified widget is not in the panel.

Parameters:
`widget` - the widget to focus
Throws:
`IllegalArgumentException` - if the specified widget is not in the panel

---

getFocus

```
public Widget getFocus()
```

Gets the widget that is the focus owner of this panel.

Returns:
the widget that own the focus on this panel or null

---

setPacked

```
public void setPacked(boolean packed)
```

Sets this panel as packed or not.
This property is used when the panel is validated: if it is packed (specified boolean is `true`), the panel is resized to the preferred size of its widgets, otherwise its widgets fill its size.

Parameters:
`packed` - `true` to pack the panel, `false` otherwise
Since:
1.0
See Also:
revalidate(), setSize(int, int)

---

isPacked

```
public boolean isPacked()
```

Gets whether this panel is packed or not.

Returns:
`true` if this panel is packed, `false` otherwise.
Since:
1.0
See Also:
setPacked(boolean)

---

invalidate

```
public void invalidate()
```

Declares that this panel needs to be laid out.

See Also:
revalidate(), validate()

---

isValid

```
public boolean isValid()
```

Gets whether this panel is valid.

A panel is valid if its contents are correctly laid out.

Returns:
`true` if this panel is valid, `false` otherwise
See Also:
invalidate(), validate()

---

revalidate

```
public void revalidate()
```

Lays out all the hierarchy of this panel.

It invalidates the panel, and then performs the method validate() asynchronously. Therefore this method is not blocked waiting until the validation of the hierarchy is done.

The panel is not validated if it is not shown or if its desktop is not shown.

Since:
1.0
See Also:
invalidate()

---

validate

```
public void validate()
```

Lays out all the hierarchy of this panel.

If the panel is already valid, it is not validated again.

The panel is repainted if it is shown on a desktop.
If the panel is declared as packed, it is resized to the preferred size of its widgets, otherwise its widgets fill its size.

See Also:
isPacked(), isValid()

---

validate

```
public void validate(int widthHint,
                     int heightHint)
```

Lays out all the hierarchy of this panel.
After this call the preferred size will have been established. The parameters defines the maximum size available for this panel, or MWT.NONE if there is no constraint.

Parameters:
`widthHint` - the width available for this panel or MWT.NONE
`heightHint` - the height available for this panel or MWT.NONE

---

handleEvent

```
public boolean handleEvent(int event)
```

Called by the system if this is the active panel and if no widget in the hierarchy of the focused widget has consumed the event. Panels handle `ej.microui.Command.UP`, `ej.microui.Command.DOWN`, `ej.microui.Command.LEFT`, and `ej.microui.Command.RIGHT` commands to manage navigation.

Specified by:
handleEvent in interface Renderable
Parameters:
`event` - the event to handle
Returns:
`true` if the panel has consumed the event, `false` otherwise

Interface Renderable

ej.mwt

All Known Implementing Classes:

Composite, Desktop, Dialog, Panel, Widget

`public interface` **`Renderable`**

A renderable is an object that is intended to be rendered on the screen.

Method Summary		Page
`int`	**computeScore** (Renderer `renderer`) Computes the score of the given renderer, normally by comparing the getStyle() of the renderable with the Renderer.getManagedStyle() of the specified renderer.	*51*
`int`	**getHeight** () Gets the height of this renderable.	*52*
Renderer	**getRenderer** () Gets the appropriate renderer for this object from the RenderingContext.	*50*
`int`	**getStyle** () Gets the style of this renderable.	*51*
`int`	**getWidth** () Gets the width of this renderable.	*51*
`int`	**getX** () Gets the x coordinate of this renderable, relative to its parent.	*51*
`int`	**getY** () Gets the y coordinate of this renderable, relative to its parent.	*51*
`boolean`	**handleEvent** (`int event`) Called by the system when an event occurred.	*52*
`boolean`	**isShown** () Gets whether or not this renderable is shown on a display.	*52*
`void`	**repaint** () Requests a repaint of this renderable.	*52*
`void`	**repaint** (`int x, int y, int width, int height`) Requests a repaint of a zone of this renderable.	*52*

Method Detail

getRenderer

Renderer **`getRenderer`**()

Gets the appropriate renderer for this object from the RenderingContext.

Returns:

the appropriate renderer or `null`

---

getStyle

```
int getStyle()
```

Gets the style of this renderable. The style is used to get the best match renderer to associate with this renderable.

Returns:
the style
Since:
1.0
See Also:
Renderer.getManagedStyle(), getRenderer()

---

computeScore

```
int computeScore(Renderer renderer)
```

Computes the score of the given renderer, normally by comparing the getStyle() of the renderable with the Renderer.getManagedStyle() of the specified renderer.
The score is bigger when the renderer matches the style and lower when it does not match.

Parameters:
`renderer` - the renderer to compute score with
Returns:
the score between the given style and the receiver style
Since:
1.0
See Also:
RenderingContext.computeScore(int, int)

---

getX

```
int getX()
```

Gets the x coordinate of this renderable, relative to its parent.

Returns:
the x coordinate

---

getY

```
int getY()
```

Gets the y coordinate of this renderable, relative to its parent.

Returns:
the y coordinate

---

getWidth

```
int getWidth()
```

Gets the width of this renderable.

Returns:
the width

---

getHeight

```
int getHeight()
```

Gets the height of this renderable.

Returns:
the height

---

isShown

```
boolean isShown()
```

Gets whether or not this renderable is shown on a display.

Returns:
`true` if the renderable is shown, `false` otherwise

---

repaint

```
void repaint()
```

Requests a repaint of this renderable.

---

repaint

```
void repaint(int x,
             int y,
             int width,
             int height)
```

Requests a repaint of a zone of this renderable.

Parameters:
`x` - the relative x coordinate of the area to repaint
`y` - the relative y coordinate of the area to repaint
`width` - the width of the area to repaint
`height` - the height of the area to repaint

---

handleEvent

```
boolean handleEvent(int event)
```

Called by the system when an event occurred.
Pointer-related events are sent to the pointed renderable. Other events are sent to the focus owner of the active panel and progress up its hierarchy while not consumed, until the desktop is reached.

Parameters:

`event` - the event to handle

Returns:

`true` if the renderable has consumed the event, `false` otherwise

Since:

0.9

Class Widget

ej.mwt

```
java.lang.Object
  └
    ej.mwt.Widget
```

All Implemented Interfaces:

Renderable

Direct Known Subclasses:

Composite

```
abstract public class Widget
extends Object
implements Renderable
```

Widget is the superclass of all the user interface objects.

There are a number of important concepts involving widgets:

Invalidation

> Whenever the state of a widget changes in a way that may affect the layout of the panel of which it is a part then the hierarchy of the widget must be ask to be revalidated. This can be achieved by invoking revalidate() on the widget.
>
> Several `Widget` methods, such as setSize(int, int), have the side-effect of asking for a revalidation of the widget. It is normal for other methods in `Widget` subclasses that affect state to do likewise.

Validation

> Validation is the process laying out the widgets on a panel. This is performed by invoking Panel.validate(), which has the side-effect of performing any required repainting after validation.
>
> The validate(int, int) method should not normally be invoked by applications or widget implementations. It is used by the `Panel` to propagate the validation request down through the widget hierarchy - widgets with children propagate the call to their children. Instead, an application will normally invoke Panel.revalidate() after making a set of changes to widgets.

Repainting

> Any widget can be asked to repaint itself by invoking repaint(). If a widget has children it will ask them to repaint. If the widget is transparent it will cause the relevant area of its parent to be repainted. Note that a repaint request does not trigger validation, and the scope of the repainting that results from a call to repaint() will never exceed the widget itself, its children (recursively), and, if it is transparent, its parent (recursively if the parent is also transparent).

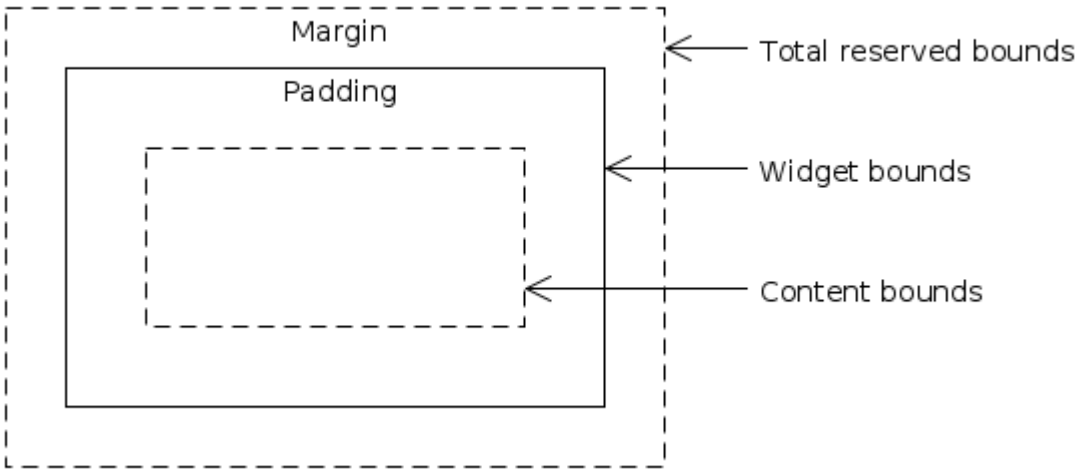
Preferred size

> A widget's preferred size is its optimal displayed size given its content and including any padding. Calling validate(int, int) causes the preferred size to be determined, normally with the assistance of a renderer. The preferred size is distinct from the widget's actual size on the display (which may itself be subject to clipping).

Actual size

> The actual size is set by calling setSize(int, int) or setBounds(int, int, int, int). Note that a widget's size never includes its margin. A Widget is rendered within a rectangle whose size is defined by the actual size.

Layout



See Also:

Renderer, WidgetRenderer

Constructor Summary	Page
Widget() Creates a new widget.	57
Widget(int x, int y, int width, int height) **Deprecated.** *use Widget() and setBounds(int, int, int, int)*	57

Method Summary		Page
protected void	**cleanRendererCache**() Cleans the renderer cache.	68
int	**computeScore**(Renderer renderer) Computes the score of the given renderer by comparing the getStyle() of the widget with the Renderer.getManagedStyle() of the specified renderer.	69
boolean	**contains**(int x, int y) Returns `true` if the (x,y) location is in the widget bounds, `false` otherwise.	62
void	**gainFocus**() Notifies the widget that it is now the focus owner of its panel and should be repainted.	64
int	**getAbsoluteX**() Returns the absolute x coordinate of the widget.	58
int	**getAbsoluteX**(int relativeX) Returns the absolute x coordinate computed from the given x coordinate relative to the widget.	61
int	**getAbsoluteY**() Returns the absolute y coordinate of the widget.	58
int	**getAbsoluteY**(int relativeY) Returns the absolute y coordinate computed from the given y coordinate relative to the widget.	61
int	**getHeight**() Returns the height of the widget.	59
Panel	**getPanel**() Gets the panel of this widget or `null` if the widget is not in a panel.	67

Composite	**getParent** () Gets the parent of this widget or `null` if the widget is not in a hierarchy.	*66*
int	**getPreferredHeight** () Returns the preferred height of the widget.	*59*
int	**getPreferredWidth** () Returns the preferred width of the widget.	*59*
int	**getRelativeX** (int absoluteX) Returns the x coordinate relative to the widget computed from the given absolute x coordinate.	*60*
int	**getRelativeY** (int absoluteY) Returns the y coordinate relative to the widget computed from the given absolute y coordinate.	*61*
Renderer	**getRenderer** () Gets the renderer associated with this widget in the rendering context.	*68*
int	**getStyle** () Gets the style of this widget.	*68*
Widget	**getWidgetAt** (int x, int y) Returns the widget at the specified location.	*63*
int	**getWidth** () Returns the width of the widget.	*58*
int	**getX** () Returns the x coordinate of the widget relative to its parent.	*58*
int	**getY** () Returns the y coordinate of the widget relative to its parent.	*58*
boolean	**handleEvent** (int event) Called by the system if the widget is the owner of the focus of the active panel.	*68*
boolean	**hasFocus** () Gets whether or not this widget is the focus owner of its panel and its panel is the active one.	*63*
boolean	**isEnable** () **Deprecated.** *use isEnabled()*	*64*
boolean	**isEnabled** () Gets whether or not this widget is enabled.	*64*
boolean	**isShown** () Gets whether or not the widget is shown on a shown panel.	*67*
boolean	**isTransparent** () Gets whether this widget is transparent or not.	*62*
boolean	**isValid** () Gets whether this widget is valid.	*66*
boolean	**isVisible** () Gets whether this widget is visible or not.	*61*
void	**lostFocus** () Notifies the widget that it is no longer the focus owner of its panel should be repainted.	*64*
void	**repaint** () Requests a repaint of this entire widget.	*67*
void	**repaint** (int x, int y, int width, int height) Requests a repaint of a zone of this widget.	*67*
void	**requestFocus** () Sets this widget as the focus owner of its panel if it is enabled.	*63*
boolean	**requestFocus** (int direction) Sets this widget as the focus owner of its panel if it is enabled.	*63*

void	**revalidate**() Lays out all the hierarchy of the panel containing this widget if one exists.	*65*
void	**setBounds**(int x, int y, int width, int height) Sets the bounds of this widget.	*60*
void	**setEnable**(boolean enable) **Deprecated.** *use setEnabled(boolean)*	*64*
void	**setEnabled**(boolean enable) Sets this widget to be enabled or not.	*65*
void	**setLocation**(int x, int y) Sets the location of this widget.	*60*
void	**setPreferredSize**(int preferredWidth, int preferredHeight) Sets the preferred size of the widget.	*59*
void	**setSize**(int width, int height) Sets the size of this widget.	*60*
void	**setTransparent**(boolean transparent) **Deprecated.** *see isTransparent()*	*62*
void	**setVisible**(boolean visible) Sets this widget visible or not.	*62*
void	**validate**(int widthHint, int heightHint) Lays out this widget if visible.	*65*

Constructor Detail

Widget

```
public Widget()
```

Creates a new widget.

By default:

- its bounds are `0`,
- it is visible,
- it is enabled.

Widget

```
public Widget(int x,
              int y,
              int width,
              int height)
```

Deprecated.

Creates a new widget specifying its bounds. Its position is relative to the position of its parent.

Parameters:

`x` - the relative x coordinate of the widget
`y` - the relative y coordinate of the widget
`width` - the width of the widget
`height` - the height of the widget

Method Detail

getAbsoluteX

```
public int getAbsoluteX()
```

Returns the absolute x coordinate of the widget.

That is, the x coordinate relative to the display.

Returns:
the absolute x coordinate of the widget

---

getAbsoluteY

```
public int getAbsoluteY()
```

Returns the absolute y coordinate of the widget.

That is, the y coordinate relative to the display.

Returns:
the absolute y coordinate of the widget

---

getX

```
public int getX()
```

Returns the x coordinate of the widget relative to its parent.

Specified by:
getX in interface Renderable
Returns:
the relative x coordinate of the widget

---

getY

```
public int getY()
```

Returns the y coordinate of the widget relative to its parent.

Specified by:
getY in interface Renderable
Returns:
the relative y coordinate of the widget

---

getWidth

```
public int getWidth()
```

Returns the width of the widget.

If the widget is not visible, its width is always equals to `0`.

Specified by:
getWidth in interface Renderable
Returns:
the width of the widget

---

getHeight

```
public int getHeight()
```

Returns the height of the widget.

If the widget is not visible, its height is always equals to `0`.

Specified by:
getHeight in interface Renderable
Returns:
the height of the widget

---

getPreferredWidth

```
public int getPreferredWidth()
```

Returns the preferred width of the widget.

The result returned is meaningful only if isValid() is `true` or if setPreferredSize(int, int) has been called explicitly.

Returns:
the preferred width of the widget

---

getPreferredHeight

```
public int getPreferredHeight()
```

Returns the preferred height of the widget.

The result returned is meaningful only if isValid() is `true` or if setPreferredSize(int, int) has been called explicitly.

Returns:
the preferred height of the widget

---

setPreferredSize

```
public void setPreferredSize(int preferredWidth,
                             int preferredHeight)
```

Sets the preferred size of the widget.

Parameters:
`preferredWidth` - the width to set
`preferredHeight` - the height to set

---

setLocation

```
public void setLocation(int x,
                        int y)
```

Sets the location of this widget.

If the widget is on a panel hierarchy, it is asked to be revalidated.

Parameters:
`x` - the x coordinate to set, relative to the parent
`y` - the y coordinate to set, relative to the parent
See Also:
revalidate()

---

setSize

```
public void setSize(int width,
                    int height)
```

Sets the size of this widget.

If the widget is on a panel hierarchy, it is asked to be revalidated.

Parameters:
`width` - the width to set
`height` - the height to set
See Also:
revalidate()

---

setBounds

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Sets the bounds of this widget.

If the widget is on a panel hierarchy, it is asked to be revalidated.

Parameters:
`x` - the x coordinate to set, relative to the parent
`y` - the y coordinate to set, relative to the parent
`width` - the width to set
`height` - the height to set
See Also:
revalidate()

---

getRelativeX

```
public int getRelativeX(int absoluteX)
```

Returns the x coordinate relative to the widget computed from the given absolute x coordinate.

Parameters:
`absoluteX` - the absolute x coordinate to convert
Returns:
the widget relative x coordinate
Since:
1.0

---

getRelativeY

```
public int getRelativeY(int absoluteY)
```

Returns the y coordinate relative to the widget computed from the given absolute y coordinate.

Parameters:
`absoluteY` - the absolute y coordinate to convert
Returns:
the widget relative y coordinate
Since:
1.0

---

getAbsoluteX

```
public int getAbsoluteX(int relativeX)
```

Returns the absolute x coordinate computed from the given x coordinate relative to the widget.

Parameters:
`relativeX` - the widget relative x coordinate to convert
Returns:
the absolute x coordinate
Since:
1.0

---

getAbsoluteY

```
public int getAbsoluteY(int relativeY)
```

Returns the absolute y coordinate computed from the given y coordinate relative to the widget.

Parameters:
`relativeY` - the widget relative y coordinate to convert
Returns:
the absolute y coordinate
Since:
1.0

---

isVisible

```
public boolean isVisible()
```

Gets whether this widget is visible or not.

Returns:
`true` if this widget is visible, `false` otherwise.
Since:
1.0
See Also:
setVisible(boolean)

---

setVisible

```
public void setVisible(boolean visible)
```

Sets this widget visible or not.

Declaring a widget as invisible means that it does not appear on the screen and its size is `(0, 0)`.

If the widget is on a panel hierarchy, it is asked to be revalidated.

Parameters:
`visible` - `true` to set this widget visible, `false` otherwise
Since:
1.0
See Also:
isValid(), revalidate()

---

setTransparent

```
public void setTransparent(boolean transparent)
```

Deprecated. *see isTransparent()*

---

isTransparent

```
public boolean isTransparent()
```

Gets whether this widget is transparent or not.

A widget is transparent if its renderer is `null`.

A transparent widget means that it will not repaint ALL the rectangular zone defined by its bounds. Then each time it needs to be repainted, its parent (recursively if also transparent) will be repainted within the bounds of the widget. Each time a non-transparent widget needs to be repainted, it is the only one to be repainted.

Returns:
`true` if this widget is transparent, `false` otherwise.
See Also:
contains(int, int)

---

contains

```
public boolean contains(int x,
                        int y)
```

Returns `true` if the (x,y) location is in the widget bounds, `false` otherwise.

The location is considered here as a relative location to parent.

Parameters:
`x` - x coordinate
`y` - y coordinate
Returns:
`true` if the (x,y) location is in widget bounds, `false` otherwise.
See Also:
isTransparent()

---

getWidgetAt

```
public Widget getWidgetAt(int x,
                          int y)
```

Returns the widget at the specified location.

If this widget does not `contains(x, y)`, `null` is returned, else this widget is returned.
The location is considered here as a relative location to parent.

Parameters:
`x` - x coordinate
`y` - y coordinate
Returns:
this widget if it `contains(x, y)`, `null` otherwise.

---

hasFocus

```
public boolean hasFocus()
```

Gets whether or not this widget is the focus owner of its panel and its panel is the active one.

Returns `false` if this widget is not on a panel.

Returns:
`true` if this widget is the focus owner, `false` otherwise
See Also:
Panel.isActive()

---

requestFocus

```
public void requestFocus()
```

Sets this widget as the focus owner of its panel if it is enabled.

If the widget is not in a panel hierarchy, nothing is done.

---

requestFocus

```
public boolean requestFocus(int direction)
```

Sets this widget as the focus owner of its panel if it is enabled.

If the widget is not in a panel hierarchy, nothing is done.
The given direction must be one of MWT.UP, MWT.DOWN, MWT.LEFT, MWT.RIGHT.
Composed widgets can override this method in order to manage internal focus.

Parameters:
`direction` - the direction followed by the focus
Returns:
`true` if the widget take the focus, `false` otherwise

---

gainFocus

```
public void gainFocus()
```

Notifies the widget that it is now the focus owner of its panel and should be repainted.

The subclasses can override this method to add behavior.

---

lostFocus

```
public void lostFocus()
```

Notifies the widget that it is no longer the focus owner of its panel should be repainted.

The subclasses can override this method to add behavior.

---

isEnable

```
public boolean isEnable()
```

Deprecated. *use isEnabled()*

---

setEnable

```
public void setEnable(boolean enable)
```

Deprecated. *use setEnabled(boolean)*

---

isEnabled

```
public boolean isEnabled()
```

Gets whether or not this widget is enabled.

A widget that is not enabled cannot get focus.
A widget that is not visible is also disabled.

Returns:
`true` if this widget is enabled, `false` otherwise

---

setEnabled

```
public void setEnabled(boolean enable)
```

Sets this widget to be enabled or not.

A widget must be enabled in order to receive focus, to be the subject of mouse focus, or to receive events.

Requests a repaint of the widget.

Parameters:
`enable` - `true` if this widget is to be enabled, `false` otherwise

See Also:
repaint()

---

revalidate

```
public void revalidate()
```

Lays out all the hierarchy of the panel containing this widget if one exists.

Since:
1.0

See Also:
Panel.revalidate()

---

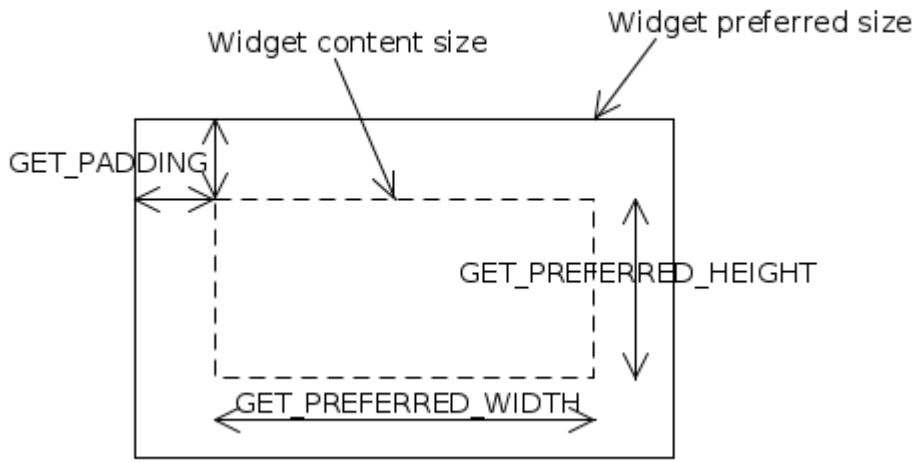
validate

```
public void validate(int widthHint,
                     int heightHint)
```

Lays out this widget if visible.

Computes the preferred size of the widget using its renderer if it is a WidgetRenderer instance:

- get the padding using Renderer.getPadding() method,
- get the preferred width using WidgetRenderer.getPreferredContentWidth(Widget) method,
- get the preferred height using WidgetRenderer.getPreferredContentHeight(Widget) method.



After this call the preferred size will have been established:

- renderer's preferred content width plus twice padding for width,
- renderer's preferred content height plus twice padding for height.

The parameters defines the maximum size available for this widget, or MWT.NONE if there is no constraint.

Parameters:
`widthHint` - the width available for this widget or MWT.NONE
`heightHint` - the height available for this widget or MWT.NONE
See Also:
isVisible(), setPreferredSize(int, int)

---

isValid

```
public boolean isValid()
```

Gets whether this widget is valid.

A widget is valid if all of these conditions are meet:

- if it is in a panel hierarchy,
- if it is visible,
- if all its panel hierarchy is valid.

Returns:
`true` if this widget is valid, `false` otherwise
See Also:
isVisible(), Panel.isValid()

---

getParent

```
public Composite getParent()
```

Gets the parent of this widget or `null` if the widget is not in a hierarchy.

Returns:
the parent of this widget or `null`

---

getPanel

```
public Panel getPanel()
```

Gets the panel of this widget or `null` if the widget is not in a panel.

Returns:
the panel of this widget or `null`

---

isShown

```
public boolean isShown()
```

Gets whether or not the widget is shown on a shown panel.

Specified by:
isShown in interface Renderable
Returns:
`true` if the widget is shown, `false` otherwise
See Also:
getPanel(), Panel.isShown()

---

repaint

```
public void repaint()
```

Requests a repaint of this entire widget. This method returns immediately; the repaint of the widget is performed asynchronously.

If the widget is not shown, nothing is done.
If the widget is transparent, it requests a repaint of its parent within the widget's bounds.

Specified by:
repaint in interface Renderable

---

repaint

```
public void repaint(int x,
                    int y,
                    int width,
                    int height)
```

Requests a repaint of a zone of this widget. This method returns immediately; the repaint of the widget is **performed asynchronously**.

If the widget is not shown, nothing is done.
If the widget is transparent, it requests a repaint of its parent within the requested bounds.

Specified by:
repaint in interface Renderable

Parameters:
`x` - the relative x coordinate of the area to repaint
`y` - the relative y coordinate of the area to repaint
`width` - the width of the area to repaint
`height` - the height of the area to repaint

---

handleEvent

```
public boolean handleEvent(int event)
```

Called by the system if the widget is the owner of the focus of the active panel.

The subclasses can override this method to add behavior.
By default, do nothing and return `false` (do not consume event).

Specified by:
handleEvent in interface Renderable
Parameters:
`event` - the event to handle
Returns:
`true` if the widget has consumed the event, `false` otherwise

---

getRenderer

```
public Renderer getRenderer()
```

Gets the renderer associated with this widget in the rendering context.

The renderer is located using the RenderingContext default algorithm.
The renderer is kept in cache until the state of the rendering context is changed.

Specified by:
getRenderer in interface Renderable
Returns:
the renderer associated with this widget, or `null` if none
See Also:
RenderingContext.getRenderer(Renderable)

---

cleanRendererCache

```
protected void cleanRendererCache()
```

Cleans the renderer cache.

Subclasses can call this method when the style of the desktop is changed and the best-fit renderer may be different (even though the pool of renderers may not have changed).
The next time getRenderer() is called a new search will be performed.

---

getStyle

```
public int getStyle()
```

Gets the style of this widget. The style is used to get the best match renderer to associate with this widget.

Always returns `0` but may be overridden in subclasses.

Specified by:
getStyle in interface Renderable
Returns:
the style
Since:
1.0
See Also:
Renderer.getManagedStyle(), getRenderer()

---

computeScore

```
public int computeScore(Renderer renderer)
```

Computes the score of the given renderer by comparing the getStyle() of the widget with the Renderer.getManagedStyle() of the specified renderer.

The score is bigger when the renderer matches the style and lower when it does not match.
The score is computed using the RenderingContext.computeScore(int, int) algorithm.

Specified by:
computeScore in interface Renderable
Parameters:
`renderer` - the renderer to compute score with
Returns:
the score between the given style and the receiver style
Since:
1.0
See Also:
RenderingContext.computeScore(int, int)

Package ej.mwt.listeners

Interface Summary		Page
FocusListener	An interface that defines notifications about widget focus.	71
PanelListener	An interface that defines notifications about changes in panel state.	72
RenderableListener	An interface that defines notifications about actions on a renderable.	74

Interface FocusListener

ej.mwt.listeners

```
public interface FocusListener
```

An interface that defines notifications about widget focus.

Method Summary		Page
void	**focusGained**(Widget widget, Widget oldWidget) Invoked when a widget gains the focus.	*71*
void	**focusLost**(Widget widget, Widget newWidget) Invoked when a widget loses the focus.	*71*

Method Detail

focusGained

```
void focusGained(Widget widget,
                 Widget oldWidget)
```

Invoked when a widget gains the focus.

Parameters:

`widget` - the widget that gained the focus
`oldWidget` - the widget that lost the focus

---

focusLost

```
void focusLost(Widget widget,
               Widget newWidget)
```

Invoked when a widget loses the focus.

Parameters:

`widget` - the widget that lost the focus
`newWidget` - the widget that gained the focus

Interface PanelListener

ej.mwt.listeners

```
public interface PanelListener
```

An interface that defines notifications about changes in panel state.

Method Summary		Page
void	**panelActivated**(Panel panel, Panel oldPanel) Invoked when a panel becomes active.	72
void	**panelDeactivated**(Panel panel, Panel newPanel) Invoked when a panel becomes inactive.	72
void	**panelHidden**(Panel panel, Desktop desktop) Invoked when a panel is hidden.	73
void	**panelShown**(Panel panel, Desktop desktop) Invoked when a panel is shown.	72

Method Detail

panelActivated

```
void panelActivated(Panel panel,
                    Panel oldPanel)
```

Invoked when a panel becomes active.

Parameters:

`panel` - the panel that became active
`oldPanel` - the panel that was active

---

panelDeactivated

```
void panelDeactivated(Panel panel,
                      Panel newPanel)
```

Invoked when a panel becomes inactive.

Parameters:

`panel` - the panel that became inactive
`newPanel` - the panel that is newly active

---

panelShown

```
void panelShown(Panel panel,
                Desktop desktop)
```

Invoked when a panel is shown.

Parameters:

`panel` - the panel that is shown
`desktop` - the desktop on which the panel is shown

---

panelHidden

```
void panelHidden(Panel panel,
                 Desktop desktop)
```

Invoked when a panel is hidden.

Parameters:

`panel` - the panel that is hidden
`desktop` - the desktop on which the panel was shown

Interface RenderableListener

ej.mwt.listeners

`public interface` **`RenderableListener`**

An interface that defines notifications about actions on a renderable.

Method Summary		Page
`void`	**renderableMoved** (Renderable `renderable`) Invoked when the renderable's location is changed.	*74*
`void`	**renderableRendered** (Renderable `renderable`) Invoked when the renderable is rendered.	*74*
`void`	**renderableResized** (Renderable `renderable`) Invoked when the renderable's size is changed.	*74*

Method Detail

renderableMoved

`void` **`renderableMoved`** (Renderable `renderable`)

Invoked when the renderable's location is changed.

Parameters:

`renderable` - the moved renderable

---

renderableResized

`void` **`renderableResized`** (Renderable `renderable`)

Invoked when the renderable's size is changed.

Parameters:

`renderable` - the resized renderable

---

renderableRendered

`void` **`renderableRendered`** (Renderable `renderable`)

Invoked when the renderable is rendered.

Parameters:

`renderable` - the rendered renderable

Package ej.mwt.migration.helper

Class Summary		Page
MigrationRenderer	**Deprecated.**	76

Class MigrationRenderer

ej.mwt.migration.helper

```
java.lang.Object
  └
    ej.mwt.rendering.Renderer
      └
        ej.mwt.rendering.WidgetRenderer
          └
            ej.mwt.migration.helper.MigrationRenderer
```

```
abstract public class MigrationRenderer
extends WidgetRenderer
```

Deprecated.

This class allow to migrate a renderer from MWT-0.9 to MWT-1.0.
Replace "`implements Renderer`" with "`extends MigrationRenderer`".

Since:
1.0

Constructor Summary	Page
MigrationRenderer()	76

Method Summary		Page
abstract int	**getHeight**(Renderable renderable) **Deprecated.** *see getPreferredContentHeight(Widget)*	77
int	**getPreferredContentHeight**(Widget widget) MICROWT-API Returns the result of getHeight(Renderable) passing the widget.	77
int	**getPreferredContentWidth**(Widget widget) MICROWT-API Returns the result of getWidth(Renderable) passing the widget.	77
abstract int	**getWidth**(Renderable renderable) **Deprecated.** *see getPreferredContentWidth(Widget)*	77
abstract int	**getZone**(Renderable renderable, int x, int y) **Deprecated.**	78

Methods inherited from class ej.mwt.rendering.Renderer
getLook, getManagedStyle, getManagedType, getMargin, getPadding, render

Constructor Detail

MigrationRenderer

```
public MigrationRenderer()
```

Method Detail

getPreferredContentWidth

`public int getPreferredContentWidth(Widget widget)`

MICROWT-API

Returns the result of getWidth(Renderable) passing the widget.

Overrides:
getPreferredContentWidth in class WidgetRenderer
Parameters:
`widget` - the widget
Returns:
the content width of the widget

---

getPreferredContentHeight

`public int getPreferredContentHeight(Widget widget)`

MICROWT-API

Returns the result of getHeight(Renderable) passing the widget.

Overrides:
getPreferredContentHeight in class WidgetRenderer
Parameters:
`widget` - the widget
Returns:
the content height of the widget

---

getWidth

`public abstract int getWidth(Renderable renderable)`

Deprecated. *see getPreferredContentWidth(Widget)*

Returns the preferred width of the specified `renderable`.

Parameters:
`renderable` - the renderable
Returns:
the width of the renderable

---

getHeight

`public abstract int getHeight(Renderable renderable)`

Deprecated. *see getPreferredContentHeight(Widget)*

Returns the preferred height of the specified `renderable`.

Parameters:
`renderable` - the renderable
Returns:
the height of the renderable

---

getZone

```
public abstract int getZone(Renderable renderable,
                            int x,
                            int y)
```

Deprecated.

Identifies the zone matching the specified coordinates in the given `renderable`.
The zones must be defined by the widget.

Parameters:
`renderable` - the renderable
`x` - the x coordinate
`y` - the y coordinate
Returns:
the identified zone

Package ej.mwt.rendering

Interface Summary		Page
Look	A look defines a set of top-level characteristics useful to the rendering, such as colors, fonts, etc.	*80*

Class Summary		Page
Renderer	A Renderer is a part of the rendering context.	*85*
RenderingContext	A rendering context is a list of Renderer instances.	*88*
Theme	A theme is a coherent set of renderers.	*92*
WidgetRenderer	This interface is the default contract between a widget and its renderer.	*95*

Interface Look

ej.mwt.rendering

```
public interface Look
```

A look defines a set of top-level characteristics useful to the rendering, such as colors, fonts, etc. A look can be associated to a theme using Theme.setLook(Look).

Since:

1.0

Field Summary		Page
int	**GET_BACKGROUND_COLOR_CONTENT** MICROWT-API The GET_BACKGROUND_COLOR_CONTENT resource type.	82
int	**GET_BACKGROUND_COLOR_DEFAULT** MICROWT-API The GET_BACKGROUND_COLOR_DEFAULT resource type.	82
int	**GET_BACKGROUND_COLOR_DISABLED** MICROWT-API The GET_BACKGROUND_COLOR_DISABLED resource type.	82
int	**GET_BACKGROUND_COLOR_FOCUSED** MICROWT-API The GET_BACKGROUND_COLOR_FOCUSED resource type.	82
int	**GET_BACKGROUND_COLOR_SELECTION** MICROWT-API The GET_BACKGROUND_COLOR_SELECTION resource type.	82
int	**GET_BORDER_COLOR_CONTENT** MICROWT-API The GET_BORDER_COLOR_CONTENT resource type.	81
int	**GET_BORDER_COLOR_DEFAULT** MICROWT-API The GET_BORDER_COLOR_DEFAULT resource type.	81
int	**GET_BORDER_COLOR_DISABLED** MICROWT-API The GET_BORDER_COLOR_DISABLED resource type.	81
int	**GET_BORDER_COLOR_FOCUSED** MICROWT-API The GET_BORDER_COLOR_FOCUSED resource type.	81
int	**GET_BORDER_COLOR_SELECTION** MICROWT-API The GET_BORDER_COLOR_SELECTION resource type.	81
int	**GET_FONT_INDEX_CONTENT** MICROWT-API The GET_FONT_INDEX_CONTENT resource type.	83
int	**GET_FONT_INDEX_DEFAULT** MICROWT-API The GET_FONT_INDEX_DEFAULT resource type.	83
int	**GET_FONT_INDEX_DISABLED** MICROWT-API The GET_FONT_INDEX_DISABLED resource type.	84
int	**GET_FONT_INDEX_FOCUSED** MICROWT-API The GET_FONT_INDEX_FOCUSED resource type.	84
int	**GET_FONT_INDEX_SELECTION** MICROWT-API The GET_FONT_INDEX_SELECTION resource type.	84
int	**GET_FOREGROUND_COLOR_CONTENT** MICROWT-API The GET_FOREGROUND_COLOR_CONTENT resource type.	83
int	**GET_FOREGROUND_COLOR_DEFAULT** MICROWT-API The GET_FOREGROUND_COLOR_DEFAULT resource type.	82
int	**GET_FOREGROUND_COLOR_DISABLED** MICROWT-API The GET_FOREGROUND_COLOR_DISABLED resource type.	83

int	**GET_FOREGROUND_COLOR_FOCUSED** MICROWT-API The GET_FOREGROUND_COLOR_FOCUSED resource type.	*83*
int	**GET_FOREGROUND_COLOR_SELECTION** MICROWT-API The GET_FOREGROUND_COLOR_SELECTION resource type.	*83*

Method Summary		Page
ej.microui.io.DisplayFont[]	**getFonts**() Gets the list of fonts of the look.	*84*
int	**getProperty**(int resource) Gets a property of the look.	*84*

Field Detail

GET_BORDER_COLOR_DEFAULT

```
public static final int GET_BORDER_COLOR_DEFAULT
```

MICROWT-API The GET_BORDER_COLOR_DEFAULT resource type.

The value `0x1` is assigned to `GET_BORDER_COLOR_DEFAULT`.

GET_BORDER_COLOR_CONTENT

```
public static final int GET_BORDER_COLOR_CONTENT
```

MICROWT-API The GET_BORDER_COLOR_CONTENT resource type.

The value `0x2` is assigned to `GET_BORDER_COLOR_CONTENT`.

GET_BORDER_COLOR_SELECTION

```
public static final int GET_BORDER_COLOR_SELECTION
```

MICROWT-API The GET_BORDER_COLOR_SELECTION resource type.

The value `0x3` is assigned to `GET_BORDER_COLOR_SELECTION`.

GET_BORDER_COLOR_FOCUSED

```
public static final int GET_BORDER_COLOR_FOCUSED
```

MICROWT-API The GET_BORDER_COLOR_FOCUSED resource type.

The value `0x4` is assigned to `GET_BORDER_COLOR_FOCUSED`.

GET_BORDER_COLOR_DISABLED

```
public static final int GET_BORDER_COLOR_DISABLED
```

MICROWT-API The GET_BORDER_COLOR_DISABLED resource type.

The value `0x5` is assigned to `GET_BORDER_COLOR_DISABLED`.

GET_BACKGROUND_COLOR_DEFAULT

`public static final int` **`GET_BACKGROUND_COLOR_DEFAULT`**

MICROWT-API The GET_BACKGROUND_COLOR_DEFAULT resource type.

The value `0x11` is assigned to `GET_BACKGROUND_COLOR_DEFAULT`.

GET_BACKGROUND_COLOR_CONTENT

`public static final int` **`GET_BACKGROUND_COLOR_CONTENT`**

MICROWT-API The GET_BACKGROUND_COLOR_CONTENT resource type.

The value `0x12` is assigned to `GET_BACKGROUND_COLOR_CONTENT`.

GET_BACKGROUND_COLOR_SELECTION

`public static final int` **`GET_BACKGROUND_COLOR_SELECTION`**

MICROWT-API The GET_BACKGROUND_COLOR_SELECTION resource type.

The value `0x13` is assigned to `GET_BACKGROUND_COLOR_SELECTION`.

GET_BACKGROUND_COLOR_FOCUSED

`public static final int` **`GET_BACKGROUND_COLOR_FOCUSED`**

MICROWT-API The GET_BACKGROUND_COLOR_FOCUSED resource type.

The value `0x14` is assigned to `GET_BACKGROUND_COLOR_FOCUSED`.

GET_BACKGROUND_COLOR_DISABLED

`public static final int` **`GET_BACKGROUND_COLOR_DISABLED`**

MICROWT-API The GET_BACKGROUND_COLOR_DISABLED resource type.

The value `0x15` is assigned to `GET_BACKGROUND_COLOR_DISABLED`.

GET_FOREGROUND_COLOR_DEFAULT

`public static final int` **`GET_FOREGROUND_COLOR_DEFAULT`**

MICROWT-API The GET_FOREGROUND_COLOR_DEFAULT resource type.

The value 0x21 is assigned to GET_FOREGROUND_COLOR_DEFAULT.

GET_FOREGROUND_COLOR_CONTENT

public static final int **GET_FOREGROUND_COLOR_CONTENT**

MICROWT-API The GET_FOREGROUND_COLOR_CONTENT resource type.

The value 0x22 is assigned to GET_FOREGROUND_COLOR_CONTENT.

GET_FOREGROUND_COLOR_SELECTION

public static final int **GET_FOREGROUND_COLOR_SELECTION**

MICROWT-API The GET_FOREGROUND_COLOR_SELECTION resource type.

The value 0x23 is assigned to GET_FOREGROUND_COLOR_SELECTION.

GET_FOREGROUND_COLOR_FOCUSED

public static final int **GET_FOREGROUND_COLOR_FOCUSED**

MICROWT-API The GET_FOREGROUND_COLOR_FOCUSED resource type.

The value 0x24 is assigned to GET_FOREGROUND_COLOR_FOCUSED.

GET_FOREGROUND_COLOR_DISABLED

public static final int **GET_FOREGROUND_COLOR_DISABLED**

MICROWT-API The GET_FOREGROUND_COLOR_DISABLED resource type.

The value 0x25 is assigned to GET_FOREGROUND_COLOR_DISABLED.

GET_FONT_INDEX_DEFAULT

public static final int **GET_FONT_INDEX_DEFAULT**

MICROWT-API The GET_FONT_INDEX_DEFAULT resource type.

The value 0x31 is assigned to GET_FONT_INDEX_DEFAULT.

GET_FONT_INDEX_CONTENT

public static final int **GET_FONT_INDEX_CONTENT**

MICROWT-API The GET_FONT_INDEX_CONTENT resource type.

The value 0x32 is assigned to GET_FONT_INDEX_CONTENT.

GET_FONT_INDEX_SELECTION

```
public static final int GET_FONT_INDEX_SELECTION
```

MICROWT-API The GET_FONT_INDEX_SELECTION resource type.

The value 0x33 is assigned to GET_FONT_INDEX_SELECTION.

GET_FONT_INDEX_FOCUSED

```
public static final int GET_FONT_INDEX_FOCUSED
```

MICROWT-API The GET_FONT_INDEX_FOCUSED resource type.

The value 0x34 is assigned to GET_FONT_INDEX_FOCUSED.

GET_FONT_INDEX_DISABLED

```
public static final int GET_FONT_INDEX_DISABLED
```

MICROWT-API The GET_FONT_INDEX_DISABLED resource type.

The value 0x35 is assigned to GET_FONT_INDEX_DISABLED.

Method Detail

getProperty

```
int getProperty(int resource)
```

Gets a property of the look.
Look defines some default resources (e.g.: GET_BACKGROUND_COLOR_DEFAULT).

Parameters:
resource - the resource to get
Returns:
the property matching the resource
Throws:
IllegalArgumentException - if the resource is not recognized

---

getFonts

```
ej.microui.io.DisplayFont[] getFonts()
```

Gets the list of fonts of the look.

Returns:
the result of the action

Class Renderer

ej.mwt.rendering

```
java.lang.Object
  └
    ej.mwt.rendering.Renderer
```

Direct Known Subclasses:
WidgetRenderer

```
abstract public class Renderer
extends Object
```

A Renderer is a part of the rendering context.
In the MVC pattern, it is the view.

It is designed to render one type of Renderable. It can perform some actions on the renderables of that type, and render them on a `ej.microui.io.GraphicsContext`.

It can be associated to a Theme.

Constructor Summary	Page
Renderer()	85

Method Summary		Page
Look	**getLook**() Gets the look of the renderer or `null` if none.	87
int	**getManagedStyle**() Gets the style managed by this renderer.	86
abstract Class	**getManagedType**() Gets the type of renderables managed by this renderer.	85
int	**getMargin**() Returns the managed objects' margin.	86
int	**getPadding**() Returns the managed objects' padding.	86
abstract void	**render**(`ej.microui.io.GraphicsContext g,` Renderable `renderable`) Renders the specified `renderable` on the given graphics context.	86

Constructor Detail

Renderer

```
public Renderer()
```

Method Detail

getManagedType

```
public abstract Class getManagedType()
```

Gets the type of renderables managed by this renderer.

Returns:
the managed objects type

---

getManagedStyle

```
public int getManagedStyle()
```

Gets the style managed by this renderer.

Always returns 0 but may be overridden in subclasses.

Returns:
the managed style
Since:
1.0
See Also:
RenderingContext.computeScore(int, int), Renderable.getStyle()

---

getMargin

```
public int getMargin()
```

Returns the managed objects' margin. The margin is the space that should be left between the bounding box of the object and others.

Always returns 0 but may be overridden in subclasses.

Returns:
the margin

---

getPadding

```
public int getPadding()
```

Returns the managed objects' padding. The padding is the space that should be left between the content of the object and its bounding box.

Always returns 0 but may be overridden in subclasses.

Returns:
the padding

---

render

```
public abstract void render(ej.microui.io.GraphicsContext g,
                            Renderable renderable)
```

Renders the specified `renderable` on the given graphics context.

Parameters:
`g` - the graphics context to be used to draw the renderable

`renderable` - the renderable to render

---

getLook

`public` Look **`getLook`**`()`

Gets the look of the renderer or `null` if none.

The look of a renderer is the one of its theme if exists.

Returns:
the look or `null`
Since:
1.0

Class RenderingContext

ej.mwt.rendering

```
java.lang.Object
  └
    ej.mwt.rendering.RenderingContext
```

```
final public class RenderingContext
extends Object
```

A rendering context is a list of Renderer instances.

This class implements the singleton pattern: there is only ever one instance of RenderingContext. The singleton is accessed via the static MWT.RenderingContext field.

Method Summary		Page
void	**add** (Theme theme) Adds a theme at the end of the list.	88
static int	**computeScore** (int style1, int style2) Computes the score between two styles.	90
Renderer	**getRenderer** (Renderable renderable) Search for the renderer that is the best match for the specified renderable.	90
Renderer[]	**getRenderers** () Gets the ordered list of renderers of the rendering context.	89
int	**getState** () Gets the current state of the rendering context.	90
void	**remove** (Theme theme) Removes a theme from the pool.	89
void	**removeAll** () Removes all the themes from the pool.	89
void	**setLook** (Look look) Sets the look of all the standard themes (Theme.isStandard() returns true) of the rendering context.	90

Method Detail

add

```
public void add (Theme theme)
```

Adds a theme at the end of the list.

The given theme is populated calling Theme.populate().

The current state is updated by this method.

Parameters:
theme - the theme to add
Throws:
NullPointerException - if the given theme is null

Since:
1.0
See Also:
getState()

---

remove

```
public void remove(Theme theme)
```

Removes a theme from the pool.

The renderers of the given theme are cleaned by calling Theme.removeAll().

If the theme is not in the pool, nothing changes.

The current state is updated by this method.

Parameters:
`theme` - the theme to remove
Throws:
`NullPointerException` - if the given theme is `null`
Since:
1.0
See Also:
getState()

---

removeAll

```
public void removeAll()
```

Removes all the themes from the pool.

The renderers of each theme are cleaned by calling Theme.removeAll().

The current state is updated by this method.

See Also:
getState()

---

getRenderers

```
public Renderer[] getRenderers()
```

Gets the ordered list of renderers of the rendering context.

The result list contains the renderers of all themes (from the first added to the last one).

Returns:
the list of renderers
Since:
1.0

---

getState

```
public int getState()
```

Gets the current state of the rendering context.
The value returned between two calls of this method:

- MUST be the same if this rendering context has not been modified,
- MUST be different if this rendering context has been modified.

The rendering context can be modified by one of these methods: add(Theme), remove(Theme), removeAll().

Returns:
the current state

---

getRenderer

```
public Renderer getRenderer(Renderable renderable)
```

Search for the renderer that is the best match for the specified renderable.

The matching is based on distance in the type hierarchy between the specified type and the type returned by Renderer.getManagedType() for each renderer in the pool. The best match is the renderer with the smallest distance.
If several renderers have the same distance, the score of each renderer is computed by the renderable. The best match is the renderer with the biggest score.
If several renderers have the same score, the first found is the best match respecting the order defined in getRenderers().

A renderer that can render type X can also be used to render objects that conform to a subtype of X.

Parameters:
`renderable` - the renderable
Returns:
the renderer that best match the specified renderable or `null` if none is found
Since:
1.0
See Also:
Renderable.computeScore(Renderer)

---

setLook

```
public void setLook(Look look)
```

Sets the look of all the standard themes (Theme.isStandard() returns `true`) of the rendering context.

Parameters:
`look` - the look to set
Since:
1.0
See Also:
Theme.isStandard(), Theme.setLook(Look)

---

computeScore

```
public static int computeScore(int style1,
                               int style2)
```

Computes the score between two styles.
It simply counts the number of matching 1-bits between the two ints.

Parameters:

`style1` - the first style
`style2` - the second style

Returns:

the score between the two styles

Since:

1.0

Class Theme

ej.mwt.rendering

```
java.lang.Object
  └
    ej.mwt.rendering.Theme
```

```
abstract public class Theme
extends Object
```

A theme is a coherent set of renderers. Typically a theme holds a set of renderers that will render a set of widgets in a consistent way.

Since:
1.0

Constructor Summary	Page
Theme() Creates a new theme.	92

Method Summary		Page
protected void	**add**(Renderer renderer) Adds a renderer at the end of the list.	93
abstract Look	**getDefaultLook**() Gets the default look of the theme.	93
Look	**getLook**() Gets the current look of the theme.	94
abstract String	**getName**() Gets the name of the theme.	93
Renderer[]	**getRenderers**() Gets the ordered list of renderers of the theme (from the first added to the last one).	94
abstract boolean	**isStandard**() Gets whether or not the renderers of the theme use only the standard resources defined in Look or not.	93
protected abstract void	**populate**() Populates all the renderers of this theme.	93
protected void	**removeAll**() Removes all the renderers from the theme.	94
void	**setLook**(Look look) Forces the look of the theme.	94

Constructor Detail

Theme

```
public Theme()
```

Creates a new theme.

Method Detail

getName

```
public abstract String getName()
```

Gets the name of the theme.

Returns:
the name of the theme

---

populate

```
protected abstract void populate()
```

Populates all the renderers of this theme.
Called by the RenderingContext when the theme is added to the context.
In the implementation of this method the renderers can be added using add(Renderer).

---

getDefaultLook

```
public abstract Look getDefaultLook()
```

Gets the default look of the theme.

Returns:
the default look

---

isStandard

```
public abstract boolean isStandard()
```

Gets whether or not the renderers of the theme use only the standard resources defined in Look or not.

When a look is set for the rendering context (using RenderingContext.setLook(Look)) that look will be set as the look (using setLook(Look)) for all themes in the context that respond `true` to isStandard().

Returns:
`true` if the renderers uses only the resources defined in Look, `false` otherwise.

---

add

```
protected void add(Renderer renderer)
```

Adds a renderer at the end of the list.

Parameters:
`renderer` - the renderer to add
Throws:
`NullPointerException` - if the given renderer is `null`
`IllegalArgumentException` - if the given renderer is already in a theme

---

removeAll

```
protected void removeAll()
```

Removes all the renderers from the theme.

---

getRenderers

```
public Renderer[] getRenderers()
```

Gets the ordered list of renderers of the theme (from the first added to the last one).

Returns:
the list of renderers

---

setLook

```
public void setLook(Look look)
```

Forces the look of the theme.

All desktops shown on a `ej.microui.io.Display` are asked to be revalidated.

Parameters:
`look` - the look to set
See Also:
Desktop.isShown(), Desktop.revalidate()

---

getLook

```
public Look getLook()
```

Gets the current look of the theme.

The current look is:

- the look set calling setLook(Look) if not `null`
- the default look of the theme

Returns:
the look
See Also:
getDefaultLook()

Class WidgetRenderer

ej.mwt.rendering

```
java.lang.Object
  └
    ej.mwt.rendering.Renderer
      └
        ej.mwt.rendering.WidgetRenderer
```

Direct Known Subclasses:

MigrationRenderer

```
abstract public class WidgetRenderer
extends Renderer
```

This interface is the default contract between a widget and its renderer. It should be implemented by all widgets' renderers.

Since:

1.0

Constructor Summary	Page
WidgetRenderer()	95

Method Summary		Page
abstract int	getPreferredContentHeight(Widget widget) MICROWT-API Returns the preferred content height of the specified renderable (excluding padding).	96
abstract int	getPreferredContentWidth(Widget widget) MICROWT-API Returns the preferred content width of the specified renderable (excluding padding).	95

Methods inherited from class ej.mwt.rendering.Renderer
getLook, getManagedStyle, getManagedType, getMargin, getPadding, render

Constructor Detail

WidgetRenderer

```
public WidgetRenderer()
```

Method Detail

getPreferredContentWidth

```
public abstract int getPreferredContentWidth(Widget widget)
```

MICROWT-API

Returns the preferred content width of the specified renderable (excluding padding).

Parameters:
`widget` - the renderable
Returns:
the content width of the renderable

---

getPreferredContentHeight

```
public abstract int getPreferredContentHeight(Widget widget)
```

MICROWT-API

Returns the preferred content height of the specified renderable (excluding padding).

Parameters:
`widget` - the renderable
Returns:
the content height of the renderable