clove - Manual

Release 0.2.1264

Josef Hahn

Jul 26, 2020

CONTENTS

CHAPTER

ONE

LICENSE

clove is written by Josef Hahn under the terms of the GPLv3 or higher.

Please read the *LICENSE* file from the package and the *Dependencies* section for included third-party stuff.

CHAPTER TWO

ABOUT

Clove is a user interface library for the web, which offers a powerful browser-side JavaScript toolkit for composing rich and neat web application frontends.

Clove is designed to get work done.

- **Integrating Clove is easy!** Just one or two *script* and one *style* to be added to your existing html before you can begin.
- **The programming interface is easy!** No esoteric tricks for simple things. It's 100% standard JavaScript, running inside the browser, with a clean api.
- **A rich and complete set of widgets is included**. This goes from labels and buttons to data views like tables and trees. It's easy to implement custom widgets.
- All widgets can be used **either stand-alone in your existing document flow or completely managed** by the powerful Clove layouting system.
- **It works in all modern web browsers**, mobile and desktop, and can help realizing applications working great in both worlds.
- Beyond widgets, Clove comes with many kind of versatile tools for typical tasks, like internationalization, branding, . . .
- It's well documented. Read the Manual for the first steps, to lookup some stuff later on, and for the api reference.
- There are live demos available, which can be opened in the web browser without any preparation needed.
- It's a true **Free Software** project without commercial pro-versions.

CHAPTER THREE

UP-TO-DATE?

Are you currently reading from another source than the homepage? Are you in doubt if that place is up-to-date? If yes, you should visit https://pseudopolis.eu/wiki/pino/projs/clove and check that. You are currently reading the manual for version 0.2.1264.

CHAPTER FOUR

MATURITY

clove is in production-stable state.

CHAPTER FIVE

DEPENDENCIES

There are external parts that are used by clove. Many thanks to the projects and all participants.

banner image, included : _meta/background.png; public domain; copied from here.

jquery, included : licensed under terms of GPLv2.

CHAPTER

SIX

INTRODUCTION

Clove is a browser-side JavaScript library which mostly provides graphical widgets. In order to use Clove, you have to include it into your web page:

```
<!doctype html>
<html>
  <head>
    ...
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript" src="clove.js"></script>
    ...
  </head>
  ...
</html>
```

The *jquery.js* can either be the included one or a different one, which is compatible. It contains the jQuery library, which is a 3rd-party component required by Clove.

There is also *clove.css* needed by Clove, but this is loaded automatically.

CHAPTER SEVEN

FIRST STEPS

After you have *included Clove in your web page*, you can use its programming interface in some way in order to support your application.

The easiest and most straight-forward way is to start with an entirely empty web page (i.e. nothing in its *body*) and to include a script, which bootstraps your application frontend:

```
<!doctype html>
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript" src="clove.js"></script>
    <script type="text/javascript">
clove.build({
    view: "MainView",
    head1: "My first Clove application",
    body: {view: "Label", label: "Hello World!"},
});
    </script>
  </head>
  <body>
  </body>
</html>
```

The interesting part is the *clove.build* call, which constructs the user interface according to a 'blueprint'. The blueprint is a JavaScript object, which contains some key/value pairs, specifying details about the user interface. In this example we see a blueprint for a *MainView*, with some properties specified (including *body*, whose value is again a blueprint for an inner widget).

The following sections will explain more details about *clove.build* and the configuration values you can specify.

Please **Note**: The pattern of bundling many configuration values into one object - typically written as *{k1:v1, … }* - and just using this object as a parameter is ubiquitous in Clove. It is not only used for widget configurations as here, but also at other places where many parameters are optional and/or vary depending on the context. **Whenever the documentation describes a parameter as a 'configuration object', or sometimes even just 'configuration', it is about such an object.** The documentation explains all relevant details about such objects for the affected functions. *This is something like 'optional parameters' for JavaScript.*

CHAPTER

EIGHT

HOW TO BUILD USER INTERFACES

In the next parts, we will see how to build a user interface in some more depth and how to use it for actual user interactions.

8.1 Building Widgets

As we have seen, user interfaces are built with the *clove.build* function. There are a few other functions, which also build user interfaces; in an equal way, but adapted for some specific situations. Directly constructing widgets by call the class constructor is not allowed. The *clove.build* call has the following general form:

```
clove.build({...}, {...});
```

The only two parameters it has are configuration objects (as mentioned above). The first one is a widget configuration (called 'blueprint' in the beginning). The second one is a build configuration, which controls some construction aspects. It is optional and not interesting in the beginning, while the first one is essential.

```
clove.build({view: 'Label'});
```

This is a very small widget configuration. Each widget configuration at least contains a *view* parameter, which specifies the widget class to instantiate. So this declaration controls what kind of widget to build.

Please **Note**: Although this is true enough for the moment, we will also see widget configurations without a *view* parameter later on. This is just because it can be implicitly clear in some situations.

Depending on the widget class, several other parameters exist. All of them are optional, but some are essential for making any real use of the widget. The *Label* widget shows just a piece of text; the *label* parameter of it specifies this text:

```
clove.build({view: "Label", label: "Hello World!"});
```

A few parameters are available for all widgets, as we will see *later on*, while many others come from the specific classes. Whenever you are interested in the properties available for a particular class, or whenever you want to learn more about any other part of the programming interface, read the relevant parts in the Developer Documentation.

For an overview of the existing widget classes, please see the *WidgetShowroom* demo. You can find the demos in your package, e.g. *clove/_meta/demos/WidgetShowroom/index.html*, or on the Clove homepage.

Please **Note**: The entire Clove programming interface resides in *clove.*, so the complete way to refer e.g. to the label class is *clove.Label*. The *view* parameter allows to omit the *clove.* though.

Some widgets are containers, which host one or more inner widgets. Those inner widgets are specified in configuration parameters as well, so a natural nesting results:

```
clove.build({
    view: "MainView",
    head1: "My first Clove application",
    body: {
        view: "Label",
        label: "Hello World!",
    },
});
```

This builds a widget of class *MainView*. This class can host one inner widget specified in *body*. This nesting can go arbitrarily deep and is used to structure complete user interfaces in one call.

8.1.1 clove.populateUI

This section closes with the introduction of a tool, which is not required but recommended to use. The clove.populateUI function enwraps time-consuming actions, as building a complex user interface can be, and gives the user some better feedback.

You should enwrap a complex clove.build call with it this way:

```
clove.populateUI(function(){
    clove.build({
        view: ...
    });
});
```

8.2 Layout And Alignment

Building complex user interfaces is possible with container widgets. Technically they are just normal widgets, but they have properties (i.e. configuration keys in a widget configuration) for child widgets. Some of them are rather special-purpose, providing some real functionality as well. Others are just intended for grouping other child widgets together and align them in some defined way.

The latter group is usually called ‘layouts’. There are a few layout types, all implementing a different but general-purpose behavior of alignment for its child widgets.

8.2.1 Stack Layout

A stack layout can be either horizontal or vertical. A horizontal layout aligns its child widgets column-based, side-by-side, all with the full height. A vertical one aligns row-based, all with the full width respectively. They are configured this way:

```
clove.build({
    view: "VerticalStack",
    rows: [
        {view: "Label", label: "Hello"},
        {view: "Label", label: "... my friend!"},
    ],
});
```

A horizontal layout is implemented by *HorizontalStack* and uses the *cols* property for its childs.

Please **Note**: There is a shortcut for horizontal and vertical stacks, which is commonly used: The *view* parameter may be omitted.

A larger example:

```
clove.build({
    rows: [
        {cols: [
            {view: "Label", label: "a1"},
            {view: "Label", label: "b1"},
            {view: "Label", label: "c1"},
        ]},
        {cols: [
            {view: "Label", label: "a2"},
            {view: "Label", label: "b2"},
            {view: "Label", label: "c2"},
        ]},
        {cols: [
            {view: "Label", label: "a3"},
            {view: "Label", label: "b3"},
            {view: "Label", label: "c3"},
        ]},
    ],
});
```

See also the *HeinersAsiaShop* demo.

8.2.2 Grid Layout

The last example tries to realize a 3x3 grid of labels. But as a grid it would not work great, because the column widths aren't connected. A real grid helps out here:

```
clove.build({
    view: "Grid",
    children: [
        {view: "Label", label: "a1", row: 0, col: 0},
        {view: "Label", label: "b1", row: 0, col: 1},
        {view: "Label", label: "c1", row: 0, col: 2},
        {view: "Label", label: "a2", row: 1, col: 0},
        {view: "Label", label: "b2", row: 1, col: 1},
        {view: "Label", label: "c2", row: 1, col: 2},
        {view: "Label", label: "a3", row: 2, col: 0},
        {view: "Label", label: "b3", row: 2, col: 1},
        {view: "Label", label: "c3", row: 2, col: 2},
    ],
});
```

8.2.3 Finetuning

There are some ways to finetune the alignment within such layouts. They make sense isolated or in some combinations in many situations.

Stretch Affinities

A layout has to divide the available space on one or both axes. In the first place this is done by asking the widgets for a preferred size. If more space is available, it can distribute it among the childs in a configurable way.

Assigning numbers to the childs' *horizontalStretchAffinity* and/or *verticalStretchAffinity* properties will lead to a distribution of free space in this exact proportions.

```
clove.build({
    cols: [
        {view: "Label", label: "Foo", horizontalStretchAffinity: 2},
        {view: "Label", label: "Bar", horizontalStretchAffinity: 1},
        {view: "Label", label: "Baz", /*horizontalStretchAffinity: 0*/},
    ],
});
```

Custom Sizes

The preferred size of child widgets is not the right choice in each situation. Sometimes it is required to set something fixed (and e.g. use scroll views if needed) in order to make the composition work. For such cases, specify *width* and/or *height* for some childs (as css length).

```
clove.build({
    cols: [
        {view: "Label", label: "Foo", width: "50pt"},
        {view: "SomethingDifferent", ..., height: "50pt"},
        {view: "Label", label: "Bar", width: "50pt"},
    ],
});
```

Stretching

Typically a widget uses all the space it can get from the layout. It bids for getting additional space by means of the *horizontalStretchAffinity*/*verticalStretchAffinity* properties. Even if it is set to 0, there is no guarantee to not get additional space.

The *strictHorizontalSizing* and *strictVerticalSizing* properties control if a widget is just placed within that space or if it actually fills the additional size.

```
clove.build({
    cols: [
        {view: "Label", label: "Foo", strictVerticalSizing: true},
        /* ... */
    ],
});
```

CHAPTER

NINE

WIDGET NAMES

After constructing a user interface, you probably will need to access some of the widgets for working with its content or state. Since you specified just the blueprint, you don't have any direct access to the constructed widgets.

The easiest way is to assign a *name* to interesting widgets and to get the actual widget instances by those names afterwards:

```
clove.build({
    cols: [
        {view: "Label", label: "Foo", name: "foolabel"},
        {view: "Label", label: "Bar", name: "barlabel"},
    ],
});
```

After executing this build call, use the clove.getByName method for getting the widget instances:

```
var foolabel = clove.getByName("foolabel");
// do something with it ...
foolabel.setLabel("Fuh");
```

9.1 Name Scopes

The model behind widget naming is called 'name scopes'. They are indeed a bit more complex that just *getByName*. There can be many separated partitions of names. This avoids name conflicts between different contexts. On the other hand, a namescope can have child scopes, which are automatically included in name lookups.

When it comes to *Custom Widgets* we will learn more, but for the moment we can use namescopes whenever we build multiple Clove user interface and want to avoid name conflicts. This is an advanced topic you can also skip at first.

For a new namescope, just construct *clove.RootNameScope* and use it in the build configuration (this is the second parameter of clove.build):

```
var mynamescope = clove.RootNameScope();
clove.build({
    rows: ...
}, {
    nameScope: mynamescope,
});
```

Each name used in the widget configuration is now known in *mynamescope*. If you would execute such code at two different places, you would end up with two isolated namescopes and no conflicts.

Getting the widgets is now possible with the namescope object:

```
var foolabel = mynamescope.getByName("foolabel");
```

CHAPTER TEN

COMMON WIDGET FUNCTIONALITY

The common base class for all widgets is *clove.Widget*. It implements some common functionality which is automatically available for each widget. All of its subclasses can be build and configured as introduced above. A later chapter will also explain how to implement *Custom Widgets*.

10.1 Widget Property System

The most essential thing all widgets have in common is the property system. It provides the foundation for all ways of working with the widgets' contents and states.

The root of this system is a key/value map of property values. The key is a simple description string like *'label'*, the value can be an arbitrary JavaScript object. This map is used for the complete configuration and state information of a widget instance.

You can access them by clove.Widget.getProperty and clove.Widget.setProperty:

```
var foolabel = clove.getByName("foolabel");
foolabel.setProperty("label", "Fuh");
```

While you can set values for arbitrary keys, some of them will be understood by the widget implementation and processed in some way. The *label* property is understood by a clove.Label, so the last example actually makes sense.

Those properties are the same as the one you set in a widget configuration, so it's the same *label* property as we already have seen in former examples (analogously you could overwrite the *rows* property of a *VerticalStack* with new child configurations) - but this way it is possible to dynamically change and retrieve those values.

For the well-known properties of a class, there is always a dedicated getter and setter as well:

```
var s = foolabel.label() + "xyz";
foolabel.setLabel(s);
```

10.2 Common Widget Properties

On top of this property system, the clove.Widget class provides some common properties (see the Developer Documentation for details):

- *visibility*: The visibility of a widget.
- *enabled*: If a widget is enabled, i.e. can interact with the user.
- *styleClass*: Additional css classes for styling. See *Styling* for more.

10.3 Common Widget Management

There are also some common management methods, i.e. for removing widgets. See the Developer Documentation for them.

CHAPTER

ELEVEN

EVENTS

The Clove event system is mostly built by the clove.Event class. It implements an event, which can be triggered from one party in certain situations and can be listened to by other parties. Events can be arbitrary things, but often have to do with user interaction like a mouse click on a certain widget.

Although not technically, there is a conceptional distinction between the owner of an event (could be a certain instance of clove.Button) and the listeners (the program components which want to react on this event). They use a clove.Event instance in different ways.

Typically event names begin with ‘On’ like in *OnClicked*.

11.1 Handling An Event

Any party can listen to a certain event instance in order to react on it. It needs to have access to the event instance and to call clove.Event.addHandler on it:

```
mybutton.OnClicked.addHandler(function(e){
    // do something
});
```

The *e* parameter is a clove.EventArgs instance, which provides some execution control and holds event-specific information.

Alternatively to this way, you can directly specify an event handler in the widget configuration:

```
clove.build({view: "Button", OnClicked: function(e){...}});
```

Please **Note**: Call clove.Event.removeHandler for unregistering the handler (or see clove.Event.addHandler for other ways).

11.2 Triggering An Event

The owner of an event triggers it when a particular situation occurs. At first it has to construct it and make it available to potential listeners somehow.

```
this.OnSomethingGreatHappened = new clove.Event();
```

Note that there is one instance per owner and that the coupling between them is loose.

The owner triggers the event with some additional information, which leads to the execution of all handlers:

```
this.OnSomethingGreatHappened.trigger({answer: 42});
```

CHAPTER

TWELVE

DATASOURCES

Datasources are an abstraction for how to get and modify a certain data store (either a real one or a virtual one computing live values). Mostly for structured views - as tables, trees, … - they provide the actual sources of data for displaying.

```
var mydatasource = new MyLottoNumberPredictionDatasource();
clove.build({view: "TableView", datasource: mydatasource});
```

The abstract base class for each datasource is clove.Datasource.

Before going deeper into the model, a ready-to-use implementation is introduced.

See also the demos about datasources.

12.1 Native JavaScript Datasources

While the datasource model is flexible enough for adapting it to arbitrary sources, there is one very simple implementation of it, which can directly be used and populated from outside. Just construct a clove.NativeDatasource instance and fill it with some data, then assign it to some view.

```
var mydatasource = new clove.NativeDatasource();
mydatasource.root().addRow(["Onion Marmelade", "10€"]);
mydatasource.root().addRow(["Rat Juice", "6.50€"]);
clove.build({view: "TableView", datasource: mydatasource});
```

The interface provides some more. Please find more details in the Developer Documentation.

12.2 Datasources Model

As mentioned before, clove.NativeDatasource is just one implementation of the base class clove.Datasource. You can implement custom subclasses and use them instead. This allows you to take the data not just from a JavaScript object, but from virtually everywhere.

In general, the data model is the following:

- There is one root node.
- Each node has a table like form with rows and columns building cells.
- Each cell has a data value; typically a String or number.
- Each cell has also is a new node.

The first three points are exactly what you would see in the spreadsheet tool of your favorite office suite, at least if you don't think too deeply what 'root node' should mean (maybe the root node is the worksheet in that analogy).

The last point might make it more difficult to imagine this construct graphically. Each cell in this table does not only have a value, but also is one more table. This can be nested arbitrarily deep.

Please **Note**: Every single node in this structure also is a table (it just could be an empty one), while each table cell is a node! Thinking ahead, this means 'node', 'table' and 'table cell' are interchangeable; which can be a confusing thing at first.

Typically you will not need the full flexibility of this model. However, the typical figures can easily be represented in such a model:

- Scalar values: Store your value in the root node.
- Lists: Store your values in the first column of the root node.
- Tables: Store your values in the rows and columns of the root node.
- Trees: Store each first-level nodes in the first column of the root node (as for lists). Then, for each of those nodes, go to the associated cell and insert the direct childs in the same way in its subtable. Do this recursively for each node until the tree is complete.

The clove.Datasource base class provides an interface which directly reflects the data structure. This must be implemented by a custom subclass, so external data consumers are able to retrieve your data structure by it. The Developer Documentation gives a full interface overview, while the following explains how a datasource works:

- clove.Datasource.getValue(ptr): Returns the value in a certain cell specified somehow by *ptr*.

For the root node, a consumer uses *undefined* as *ptr*. How any inner node, it uses another datasource function to get a pointer to it:

- clove.Datasource.valuePointer(irow, icol, parent): Returns a clove.DatasourceValuePointer, pointing to a non-root node inside the datasource. It is specified by a row and column index and a parent node. If the parent node is not the root node, you would use the same method for getting a pointer to it. This nested calls directly reflect the nested data structure.

Example:

```
mydatasource.getValue(
    mydatasource.valuePointer(1, 3,
        mydatasource.valuePointer(3, 7)
    )
);
```

This returns the value of cell 1/3 of the node in cell 3/7 of the root node.

12.2.1 Implementing A Custom Datasource

The already mentioned methods and some more must be implemented in a datasource. The first steps are again about the clove.DatasourceValuePointer class.

- clove.Datasource.valuePointer(irow, icol, parent): This method constructs and returns such a pointer:

```
valuePointer(irow, icol, parent) {
    //...
    return new clove.DatasourceValuePointer(irow, icol, backendObject);
}
```

The major trick here is the *backendObject*. A backend object for a node is an object, which is opaque for Clove and for consumers, but which contains everything your custom datasource implementation needs to answer data queries to that node. This could be custom index structures, references, or direct parts of your custom data structure.

When this function is called with *parent*:=`undefined`, you have to use an internal representation of the *irow*/*icol* cell in the root as *backendObject*. Otherwise you have to use the representation of an inner structure. For finding the right path in your structure, you must take care of the backend object you can get from *parent*:

```
var myRootFoo = new Foo(); // our external source
//...

class MyDatasource extends clove.Datasource(Object) {

    valuePointer(irow, icol, parent) {
        if (parent)
            var parentfoo = parent.backendObject;
        else
            var parentfoo = myRootFoo;
        return new clove.DatasourceValuePointer(irow, icol, parentfoo.
↪getInnerFoo(irow, icol));
    }

    //...
```

- clove.Datasource.getValue(ptr): Return the value for the specified node:

```
getValue(ptr) {
    return ptr.backendObject.getFooValue();
}
```

This alone should be enough to fetch data from an arbitrary place in your structure. There are some more parts required though:

- clove.Datasource.rowCount(parent) and clove.Datasource.columnCount(parent): For a given parent node, return how much rows and columns it has:

```
rowCount(parent) {
    return parent.backendObject.getInnerFooRowCount();
}

// same for columnCount
```

- clove.Datasource.parent(ptr): Return the parent for the node given as clove.DatasourceValuePointer *ptr*:

```
parent(ptr) {
    var pfoo = ptr.backendObject.getParentFoo();
    if (pfoo == myRootFoo)
        return undefined;
    else
        return new clove.DatasourceValuePointer(pfoo.rowindex(), pfoo.columnindex(),␣
↪pfoo);
}
```

The last essential thing is to notify consumers whenever something changed in your data structure. Otherwise you would never - or only randomly - get updated views. It could also fail when it assumes some row or column counts while some of them disappeared meanwhile in your structure.

In order to correctly notify consumers, some events from your clove.Datasource instance must be triggered:

```
this.OnDataInsert.trigger({r1: ..., r2: ..., c1: ..., c2: ..., parent: ...});

this.OnDataRemove.trigger(/* as above */);

this.OnDataUpdate.trigger(/* as above */);
```

Those events are for creation, removal and updating of nodes. The parameters are *parent*, which is a pointer as described above, and first/last row and column index which is affected by the event.

12.3 Headersources

There is an additional concept which is about the headers of rows and columns. In order to configure them (mostly the header text), some views also accept a headersource. This is a different interface, potentially implemented by a different object, which provides those information for entire rows and columns. However, the interface works similar and very often both interface are implemented by the same object.

Please read about clove.Headersource for details.

Please **Note**: The clove.NativeDatasource implementation introduced above also implements this interface and also provides methods for configuring the headers from outside.

12.4 Data Bindings

While datasources are the only way to display data in clove.DataView widgets, they are also part of the powerful data binding foundation. This allows to bind a datasource to arbitrary properties of arbitrary widgets.

For binding a node in a clove.Datasource to a property of a widget, a clove.DataBinding is to be used. This should be created with the clove.databind function and

- either connected in a widget configuration:

```
clove.build({view: "EditBox", text: clove.databind({datasource: mydatasource})});
```

- or later on with clove.Widget.bindProperty:

```
mywidget.bindProperty("text", clove.databind({datasource: mydatasource}));
```

Read about clove.databind for more options.

12.5 Asynchronous Data Sources

Asynchronous data sources are implemented by clove.AsyncDatasource. There are some interesting subclasses like clove.AjaxAsyncDatasource. Read the api documentation for details.

CHAPTER

THIRTEEN

DIALOGS

A dialog is a body area with a title bar, floating above the rest of the interface (looks and feels like dialogs in desktop environments). Be aware that they are simulations which run inside the main browser window. Opening an actual popup or a new browser tab is a different story.

There are some ways to create dialogs, with different simplicity levels which of course also differ in power.

See also the *HeinersAsiaShop* demo.

13.1 Simple Dialogs

Message dialogs with clove.messageDialog can show a message text in a dialog and requests the user to press a button. This dialog is modal, so the interface behind is not reachable. When the user has clicked on a button, the dialog is closed and an event is triggered. Example:

```
var OnProceed = clove.messageDialog({
    message: "Hello World!",
    buttons: ["Foo", "Bar", "Baz"],
});
OnProceed.addHandler(function(e) {
    console.log("Clicked on #" + e.button);
});
```

A similar dialog with a input text box is provided by clove.inputDialog. A slightly more complicated variant with a custom inner part (but including the button bar) is provided by clove.conversationDialog. Read about them for details.

13.2 Complex Dialogs

The widget class clove.Dialog implements the container with the frame and title bar, used to be the dialog. Like other containers, it has a property for an inner widget. But the usage it typically different, because direct usage in clove.build would lead either to a full-screen dialog or to some dialog-like looking box within your main interface. Both is obviously not intended behavior for a dialog.

Instead, use clove.Dialog.show, mostly in the same way as clove.build, and build an arbitrary inner interface for this dialog:

```
var dlg = clove.Dialog.show({
    view: "Dialog",
    title: "My Dialog",
    body: {view: "Label", label: "Hello World!"},
});
```

Although not for clove.build, the return value of clove.Dialog.show is important to keep. Everything you build with this function is not part of the root *namescope*. Instead, you must use *dlg.namescope* for getting widgets by name.

The clove.Dialog widget itself is also available in this result as *dlg.widget*. This is particularly useful for eventually closing the dialog later on with *dlg.widget.close();*.

13.3 Popups

Even more generic is to use clove.utils.popup. This opens any kind of widget floating in the same way as a dialog would do. Read about it for details.

CHAPTER
FOURTEEN

INTERNATIONALIZATION

Internationalization of a user interface has many facets. Some of them should be handled fine directly by the browser (this is true e.g. for string representations for numbers, dates and times). String translation is the next big topic and is what Clove helps you with.

There is even more, like left-to-right vs. right-to-left text flow. But those are beyond the scope of this manual.

The Clove Internationalization support is implemented in the clove.I18N class. There exists a single instance of this class as *clove.i18n*.

The idea behind translating a user interface is simple:

- Collect all your interface strings in a table, translated into each target language:

```
clove.i18n.addString('HelloWorld', {en:"Hello world", de:"Hallo Welt", nl:"Hallo␣
↪wereld"});
clove.i18n.addString('ThanksForVisiting', {en:"Thank you for v...
```

- Take strings from this object whenever a user interface text is build:

```
foolabel.setLabel( clove.i18n.HelloWorld );
```

This automatically does the translation to the language set in the user's system settings.

CHAPTER FIFTEEN

CUSTOM WIDGETS

For complex and dynamic interfaces it might be a good idea to encapsulate some isolated parts to a new custom widget. Another reason for a custom widget could be to implement some entirely new behavior or functionality.

A custom widget class . . .

- either directly or indirectly inherits from clove.Widget
- provides a constructor with the same signature as clove.Widget
- overrides some of the methods, depending on what it should do
- often introduces new *properties*
- virtually always implements adding some content to the html dom tree and working with it

A rather minimalist widget implementation is the following:

```
class MyHelloWorldLabel extends clove.Widget {

    constructor(config, domnode) {
        super(config, domnode);
        // this.contentnode is our html dom node
        this.contentnode.textContent = "Hello World";
    }

    // optional
    doinit() {
        // initialization steps ...
    }

}
```

Once this class is defined, you can use it in widget configurations:

```
clove.build({view: "MyHelloWorldLabel"});
```

For all kinds of graphical representation, you should use css as often as you can and especially follow the *Styling* section.

The *config* object is the widget configuration like it is passed in at the clove.build call. However, on the way down the constructor chain, this object can (and typically will) get changed. clove.utils.applyDefaultsToConfig is a typical tool in this context.

15.1 Widget Properties

Instead of the static text from the example above, you often want to provide a way to let the widget consumer decide about content or behavior. The widget property system, which does exactly this, was already introducted above from the consumer perspective. For the widget developer, providing a property *myproperty* means the following:

- Calling *this.declareProperty("myproperty", mydefaultvalue)* in the constructor. The default value is optional.
- Optionally provide *myproperty_getter* and/or *myproperty_setter* for dynamically computing property data or for reacting on changes, e.g. by changing stuff in the html content.

15.2 Layouts And Sizing

Widgets have to live together on a more or less large area, where they get a position and size. With the size actually allocated for the widget, the internal routines then have to align the own content in a proper way. The custom widget has to play together with these mechanisms. This can happen more or less complicated, as the following sections describe.

But beforehand, please note: clove.Widget comes with an existing implementation for all this sizing matters, which works for some simpler cases of html content. It will however fail whenever the content does some internal positioning or uses block elements and in many other cases.

15.2.1 Using An Existing Layout Implementation

A very easy and powerful way to build new widgets is to compose it of existing widgets and use a layout internally, instead of manually generating new html content. Obviously this does only work if the existing widgets are enough to build your custom one.

Implement such a composed widget by adding one of the layout mixings to the superclass chain and directly give it a configuration:

```
class MyLabelComposition extends clove.StackLayout(clove.Widget) {

    constructor(config, domnode) {
        // add some more stuff to (a copy of) the config
        config = clove.utils.applyDefaultsToConfig(config, {
            orientation: "vertical",
            children: [
                {view: "Label", label: "Foo"},
                {view: "Label", label: "Bar"},
            ],
        });
        super(config, domnode);
    }

}
```

15.2.2 Provide Custom Sizing Support

If the custom widget manages real own html content, it typically has to implement the following methods for sizing support:

- clove.Widget.computeMinimalWidth, clove.Widget.computePreferredWidth, clove.Widget.computeMinimalHeightForWidth, clove.Widget.computePreferredHeightForWidth: Return minimal and preferred sizes on both axes for the widget in its current state as pixel values.
- clove.Widget.doresize: Realign the internal stuff properly into the current actual widget size.

Whenever the some relevant parts of the widget state changed, it has to call clove.Widget.relayout in order to refresh its sizes.

15.2.3 Prevent Name Conflicts

When you implement a new custom widget by somehow composing it of existing ones, you typically assign names to some internal parts, so you can access and work with them internally later on. But there is a pitfall: Those names would conflict with other ones outside of the widgets, or even with a second instance of the same custom widget. This can be solved this way:

- Place your custom widget in a new namescope. This can be done by adding clove.NameScope to your inheritance chain, so you might end up with a superclass like *clove.NameScope(clove.StackLayout(clove.Widget))*. This leads to an isolated namescope for all your inner widgets.
- If your container isn't a container, you are done. Otherwise there is a new problem now: When a consumer uses the new custom widget, placing also widgets in this container, and tries to access those ones by name, it will fail. This is because those widgets aren't in the namescope you would expect, but they are in that isolated new one. The solution is to introduce one more namescope, just for this containing part, and add just this one as child namescope to the original one.

```
class MyContainerWidget extends clove.NameScope(clove.StackLayout(clove.Widget)) {

    constructor(config, domnode) {
        super(clove.utils.applyDefaultsToConfig(config, {
            children: [
                ...,
                {rows: [], name: "innercontainer", newNameScope: true},
            ],
        }), domnode);
        this.declareProperty("body", {rows: []});
    }

    body_setter(v) {
        this.getByName("innercontainer").setChildren([v]);
    }

    doinit() {
        super.doinit.call(this);
        this.containingNameScope().addChildNameScope(this.getByName("scroll"));
    }

}
```

15.3 Best Practices

There are some guidelines for developing custom widgets, which should be known.

Add a css class to the top node

There is a another html dom node in each widget, *this.topnode*, which encloses the actual content node. This has different technical purposes and is also typically used to assign a css class to, which represents the widget class. This allows to style the widget parts.

```
class MyFooWidget extends clove.Widget {

        constructor(config, domnode) {
            super(config, domnode);
            $(this.topnode).addClass("myfoowidget");
            //...
```

clove.utils.suspendResizing

When a program logic does large changes on some user interface parts, this can be time consuming. Think about adding a large bunch of data to a widget. Depending on how it is designed, you might end up with calling something like *somewidget.addData(something);* lots of times. Each of this call likely will trigger the recomputation of the widget sizing. Most of those computations - all but the last to be exact - have no value at all, but can take a considerable amount of time.

For large computations within your widget routines, you should temporarily suspend the resizing.

```
var xxx = clove.utils.suspendResizing();
try {
    //...
}
finally {
    clove.utils.resumeResizing(xxx);
}
```

CHAPTER

SIXTEEN

STYLING

Styling in Clove should be done entirely with css classes. There are only rare cases where directly assigning styles to elements is the better way. Since a Clove widget should be represented by a css class as well, styling is natural.

```css
.myfoowidget {
    color: blue;
}
```

More classes can be added at some place of the dom node for supporting finer styles.

Also note the dom structure each widget has: parent > topnode > contentnode > content. The topnode is the root node of a widget. Within it, there is the content node and within it the actual content. For some css selectors this is important to know.

```css
.myfoowidget > div > a {
    color: blue;
}
```

This would colour each *a* node directly in the content node of a MyFooWidget.

16.1 Clove Common Styles

There are some common style classes. They help for easily styling some stuff in default situations, like larger control appearance, important texts, errors texts or margins. It's not required to use them, but they avoid some typing and makes application styling easier to adapt.

For a list of existing common classes, inspect code samples or *clove.css* and find css class names beginning with *clovestyle_*. Those kinds of classes are particularly interesting:

- Text styling: *clovestyle_text_**
- Panels (i.e. container areas for some content): *clovestyle_panel_**
- Margins around arbitrary widgets: *clovestyle_*margin**

CHAPTER

SEVENTEEN

USING WIDGETS IN EXISTING WEBPAGES

The typical way is to build widgets completely in a Clove context. However, it is also possible to build widgets into existing parts of an existing webpage. This way reuses your existing page but just adds some inner parts to it.

At first you have to get the html dom node which shall be populated by a widget. Then, use this in the build configuration of the clove.build call:

```
var mydiv = ...;
clove.build({
    rows: ...
}, {
    domnode: mydiv,
});
```

You might also set the clove.Widget.doStandaloneResizing property on such a widget in order to make it automatically resize itself.

CHAPTER

EIGHTEEN

API REFERENCE

18.1 Class list

18.1.1 Class clove

`class clove`

Clove root namespace.

Public Functions

`build`(config, buildconfig)

Constructs new widgets according to a widget configuration and controlled by the build configuration.

Read the Manual for details.

Parameters

- `config`: The widget configuration; like a blueprint for the new widget.
- `buildconfig`: The build configuration; specifies some aspects about the construction.

`getByName`(name)

Finds a widget by name in the clove root namescope.

Read the Manual for details.

Parameters

- `name`: The widget name.

`populateUI`(initfct, populateconfig)

Utility for user interface creation.

Using this function for building the main user interface (or other complex pieces) enhances the performance and can provide a loading indicator visible to the user.

Parameters

- `initfct`: A function which actually builds the user interface.
- `populateconfig`: A (optional) configuration object which specifies some behavior aspects during initialization.

`conversationDialog`(config)

Shows a conversation dialog (small and easy dialog box).

This builds a small *clove::Dialog* with an arbitrary body widget and a button bar. It returns a dialog result structure, as *clove::utils::popup()*, also containing `OnProceed`. This *clove::Event* triggers when the user clicks on one of the buttons in the button bar.

config may contain:

- `title`: *Dialog* title text.
- `body`: Inner widget as widget configuration like for *clove::build()*.
- `icon`: *Icon* as *clove::Icon*.
- `buttons`: List of button label strings.

  Parameters

  - `config`: A conversation dialog configuration object.

`messageDialog`(config)

Shows a message dialog.

This builds a small *clove::Dialog* which contains a message and asks the user to click on a button. It returns an *clove::Event* which is triggered when the user clicked on a button.

config may contain:

- `title`: *Dialog* title text.
- `message`: The message text.
- `buttons`: List of button label strings.

Note: In very exotic situations, it might be useful to access the complete dialog result structure as *clove::conversationDialog()* would return. This is possible with the `dlgres` property of the returned object.

Parameters

- `config`: A conversation dialog configuration object.

`inputDialog`(config)

Shows an input dialog.

This builds a small *clove::Dialog* which asks the user for some text input. It returns an *clove::Event* which is triggered when the user finished.

config may contain:

- `title`: *Dialog* title text.
- `question`: The question text.
- `buttons`: List of button label strings.
- `defaultAnswer`: The initial answer string.

Note: In very exotic situations, it might be useful to access the complete dialog result structure as *clove::conversationDialog()* would return. This is possible with the `dlgres` property of the returned object.

Parameters

- `config`: A conversation dialog configuration object.

`selectionDialog`(config)

Shows a dialog with a list of choices.

This builds a small *clove::Dialog* which asks the user to choose an item from a list. It returns an *clove::Event* which is triggered when the user finished.

config may contain:

- `title`: *Dialog* title text.
- `question`: The question text.
- `buttons`: List of button label strings.
- `defaultAnswer`: The initial answer string.
- `choices`: The list of choice strings.

Note: In very exotic situations, it might be useful to access the complete dialog result structure as *clove::conversationDialog()* would return. This is possible with the `dlgres` property of the returned object.

Parameters

- `config`: A conversation dialog configuration object.

`Visible`

Visibility value for a visible widget.

`Invisible`

Visibility value for a widget, which is invisible but occupies its room.

`InvisibleCollapsed`

Visibility value for a widget, which is not visible and does not take any space.

`i18n`

Provides internationalization features.

Read the Manual for details.

`rootNameScope`

The Clove root namescope.

`browser`

Some data helpful for browser detection.

`databind`(config)

Constructs a *clove.DataBinding*.

The returned binding is for usage in a widget configuration. It is not completely configured, so it will not work out of the box in other situations.

The databind configuration may contain the following parameters:

- `datasource`: The *clove.Datasource* instance to bind.
- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.
- `dataDirection`: One of ‘todatasource’, ‘towidget’ or ‘bidirectional’.
- `valueComparator`: A `function(a,b)` which decides if two values are considered as equal. Default is something like `a==b`, but also looks for an `equals` method on the objects.

  Parameters

  - `config`: The databind configuration.

- `valueConverter`: A *clove::DataBindingConverter* which translates between datasource value and user interface representation. Default is a noop conversion.

MenuSeparator

A menu separator for usage in *clove::AbstractMenu::actions()*.

notifications

The clove notification controller.

It is an instance of *clove::NotificationController*.

class AbstractMenu : **public** *clove*::*Widget*

A base class for widgets which present a menu of actions to the user.

Subclassed by *clove::Menubar*, *clove::PopupMenu*

Public Functions

actions()

The actions to be shown in this menu.

It is a list of action configuration objects, each having a structure like `{name:'myaction', label:'My menu action'}. It can have a list of sub-items in the subactions` property.

Instead of a simple list, it may also be a *clove::Datasource* with the action configuration structures aligned in rows.

Use *clove::MenuSeparator* for a separator.

One action allows this properties:

- `name`: The action name.
- `label`: The label string.
- `icon`: A *clove::Icon*.
- `disabled`: If it is disabled.
- `invisible`: If it is invisible.
- `checkable`: If it is checkable.
- `checked`: If it is checked.

setActions(value)

Setter for *actions()*.

Parameters

- `value`: The new value.

OnActionTriggered

Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

OnBeforeSubactionsExpanded

Triggered just before a branch of subaction is expanded.

Implement this event for populating it dynamically.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch``()`

Alias for *horizontalStretchAffinity()*.

`setHstretch``(value)`

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy``()`

If the widget is in busy state, typically resulting in a loading animation.

`setBusy``(value)`

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy``()`

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy``(token)`

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus``()`

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize``()`

Returns inner width and height of this widget.

`outerSize``()`

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class AjaxAsyncDatasource` : **`public`** *clove*::*AsyncDatasource*, **`public`** *clove*::*Headersource*

A *clove::AsyncDatasource* implementation which uses Ajax requests for data retrieval and modification.

Public Functions

`AjaxAsyncDatasource`(config)

The ajax configuration may contain:

- `data_pullUrl`: The url for data retrieval.
- `data_pushUrl`: The url for data modification.
- `data_pullParams`: Additional parameters for data retrieval.
- `data_pushParams`: Additional parameters for data modification.
- `data_params`: Additional parameters.
- `data_mayHaveChildren`: If the data items might have children.
- `data_autoPullCharily`: If automatic data fetching should be less aggressive.
- `params`: Additional parameters.
- `rootType`: The root type. This controls the prefix of the url and parameter keys to consider
- `ajaxAdditionalArgs`: Additional arguments for the ajax request.
- `answerIdKeyName`: The key name for the identification column used in reading server answers.
- `requestIdKeyName`: The key name for the identification used in requests.
- `headerConfigs`: Additional static header configurations. for fetching the first (and often only) level of objects. Instead of `data_`, other prefixes are allowed as well. The return type name of an retrieved object controls which prefix to use for fetching child objects.

  Parameters
  - `config`: The Ajax configuration.

`OnDataArrived`

Triggered when new data arrived from the server.

This is a *clove.Event* instance. See Manual for details.

`getAjaxId`(ptr)

Returns the Ajax id for a data cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

`do_async_pull`(ptr, requestConfig)

Executes an asynchronous data modification.

Do not call this from outside, use *clove::AsyncDatasource::async_pull()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

do_async_push (ptr, v, requestConfig)

Executes an asynchronous data retrieval.

Do not call this from outside, use *clove::AsyncDatasource::async_push()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

async_pull (ptr, requestConfig)

Asynchronously modifies a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_pull()* instead.

`requestConfig` may contain:

- `onsuccess`: A `function()` called when the operation is completed and reflected in this datasource.
- `onfailed`: A `function(error)` called when an error occurred.
- `reason`: An optional custom object which can be used by observers for custom decisions.

  Parameters

  - `ptr`: The *clove::DatasourceValuePointer*.
  - `requestConfig`: The request configuration.

async_push (ptr, v, requestConfig)

Asynchronously retrieves a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_push()* instead.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

clearCache ()

Clears the internal caches.

getValue (ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata (ptr)

Returns an object of metadata information (like icons, status infos used for styling, …). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

changeValue (ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

`rowCount` (*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`columnCount` (*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`valuePointer` (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent` (ptr)

Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

`valuePointerNavigateInDepth` (ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

`OnDataInsert`

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`

Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)

Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader`(irow, *parent*)

Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible`(*parent*)

If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible`(*parent*)

If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

`class AsyncDatasource` : **`public`** *clove*::*Datasource*, **`public`** *clove*::*Headersource*

A *clove::Datasource* implementation which supports asynchronous data operations like Ajax requests.

Subclassed by *clove::AjaxAsyncDatasource*

Public Functions

`AsyncDatasource`(config)

The configuration may contain:

- `'errorhandler'`: Optional `function(error)` as fallback error handler.
- `'cacheAge'`: The maximum cache age in seconds.

  Parameters
  - `config`: The async datasource configuration.

`do_async_pull`(ptr, requestConfig)

Executes an asynchronous data modification.

Do not call this from outside, use *clove::AsyncDatasource::async_pull()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

`do_async_push`(ptr, v, requestConfig)

Executes an asynchronous data retrieval.

Do not call this from outside, use *clove::AsyncDatasource::async_push()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

`async_pull`(ptr, requestConfig)

Asynchronously modifies a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_pull()* instead.

`requestConfig` may contain:

- `onsuccess`: A `function()` called when the operation is completed and reflected in this datasource.
- `onfailed`: A `function(error)` called when an error occurred.
- `reason`: An optional custom object which can be used by observers for custom decisions.

  Parameters
  - `ptr`: The *clove::DatasourceValuePointer*.
  - `requestConfig`: The request configuration.

`async_push`(ptr, v, requestConfig)

Asynchronously retrieves a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_push()* instead.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

`clearCache`()

Clears the internal caches.

getValue (ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata (ptr)

Returns an object of metadata information (like icons, status infos used for styling, ...). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

changeValue (ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount (*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount (*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

parent (ptr)

Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

valuePointerNavigateInDepth (ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

OnDataInsert

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataRemove

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`
Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)
Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader` (irow, *parent*)
Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible` (*parent*)
If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible` (*parent*)
If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`
Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`
Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`
Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`
Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

`class AudioPlayer` : **`public`** *clove*::*MediaPlayer*
A widget which plays an audio source and optionally allows user controls.

Public Functions

`static` **`canPlayType`**`(t)`

Determines if the browser can play a certain audio type and returns a string as described in the html specs (->”canPlayType”).

Parameters

- `t`: A mimetype string.

`autoPlay``()`

If to automatically start playback as soon as possible.

`setAutoPlay``(value)`

Setter for *autoPlay()*.

Parameters

- `value`: The new value.

`showControls``()`

If to show user controls for play, pause and more.

`setShowControls``(value)`

Setter for *showControls()*.

Parameters

- `value`: The new value.

`currentMediaPosition``()`

The current media playback position in seconds.

`setCurrentMediaPosition``(value)`

Setter for *currentMediaPosition()*.

Parameters

- `value`: The new value.

`loop``()`

If to playback in an endless loop.

`setLoop``(value)`

Setter for *loop()*.

Parameters

- `value`: The new value.

`muted``()`

If to mute the audio playback.

`setMuted``(value)`

Setter for *muted()*.

Parameters

- `value`: The new value.

`preload``()`

If to begin loading the media data as soon as possible.

`setPreload``(value)`

Setter for *preload()*.

Parameters

- `value`: The new value.

`volume``()`

The audio playback volume between `0.0` and `1.0`.

`setVolume``(value)`

Setter for *volume()*.

Parameters

- `value`: The new value.

`source``()`

The media source URL.

`setSource``(value)`

Setter for *source()*.

Parameters

- `value`: The new value.

`duration``()`

The duration of the loaded media source in seconds.

`hasEnded``()`

If the playback has ended (i.e. reached the end).

`isPaused``()`

If the playback is logically paused.

This does not return `true` just when loading stalls.

`nativeNode``()`

Returns the native html dom node.

`play``()`

Starts playback.

`pause``()`

Pauses playback.

`OnLoadingAborted`

Triggered when the media loading aborted for some reasons (e.g. network errors).

This is a *clove.Event* instance. See Manual for details.

`OnReadyForPlayback`

Triggered when there are enough data for starting playback.

This is a *clove.Event* instance. See Manual for details.

`OnDurationChanged`

Triggered when the playback duration changed.

See also *duration()*.

This is a *clove.Event* instance. See Manual for details.

`OnHasEnded`

Triggered when the playback has ended.

See also *hasEnded()*.

This is a *clove.Event* instance. See Manual for details.

`OnIsPaused`

Triggered when the playback logically paused.

This is not triggered when loading stalls.

This is a *clove.Event* instance. See Manual for details.

`OnIsPlaying`

Triggered when the playback logically starts.

This is not triggered when loading has stalled and resumes.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

`getPreferredHeightForWidth``(w)`
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

`addStyleClass`(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

`removeStyleClass`(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

`isStyleClass`(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope``()`
The namescope this widget contains to.

`nameScope``()`
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters
- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters
- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters
- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters
- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Border` : **`public`** *clove*::*Widget*
A single-widget container framing the inner one with (a css-styled) border.

Public Functions

`body`()
The inner widget as widget configuration like for *clove::build()*.

`setBody`(value)
Setter for *body()*.

Parameters
- `value`: The new value.

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)
Setter for *name()*.

Parameters
- `value`: The new value.

`enabled`()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

`styleClass`()
Custom css class(es).

`setStyleClass`(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

`style`()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)
Setter for *style()*.

Parameters
- `value`: The new value.

`visibility`()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Button` : **`public`** *clove*::*Widget*

A button allows the user to trigger a particular program event.

Subclassed by *clove::PopupMenuButton*

Public Functions

`label`()

The label text.

`setLabel`(value)

Setter for *label()*.

Parameters

- `value`: The new value.

`icon`()

The optional *clove::Icon* button icon.

`setIcon`(value)

Setter for *icon()*.

Parameters

- `value`: The new value.

`checkable`()

If the button is checkable (i.e. has a checked-flag).

If it has, the checked-flag is controlled by *checked()*.

`setCheckable`(value)

Setter for *checkable()*.

Parameters

- `value`: The new value.

`checked`()

For a checkable button, return if it is checked.

See also *checkable()*.

User interactions do not toggle the checked flag by default. This must be scripted in own event handlers as needed.

`setChecked`(value)

Setter for *checked()*.

Parameters

- `value`: The new value.

`OnClicked`
Triggered when the user clicks on this button.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

`setProperty`(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.
Parameters
- `rootNameScope`: The root namescope to add this widget to.

`doinit`()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Carousel` : **`public`** *clove*::*TabView*

A carousel container.

Something like a tab bar, but automatically switching forward tabs in regular intervals (and with a different style).

Public Functions

`interval`()

The tab switching interval (in milliseconds).

`setInterval`(value)

Setter for *interval()*.

Parameters

- `value`: The new value.

`play`()

Begins automatical tab switching.

`pause`()

Stops automatic tab switching.

`isPlaying`()

If automatic tab switching is enabled.

`next`()

Switches to the next tab.

`tabs`()

The list of tabs.

Each tab is a widget configuration, like for *clove::build()*, with some additional optional keys which holds infos like the tab header text. So, for example, one item in that list could be `{view:'Something', ..., tabLabel:'Tab1'}`. See also *addTab()*.

setTabs (value)
Setter for *tabs()*.

Parameters
- `value`: The new value.

currentTab ()
The index of the currently selected tab.

setCurrentTab (value)
Setter for *currentTab()*.

Parameters
- `value`: The new value.

userMayAddTabs ()
If to show a button for adding new tabs.

If `true`, implement a handler for OnTabCreationRequested and create a tab with *addTab()* there.

setUserMayAddTabs (value)
Setter for *userMayAddTabs()*.

Parameters
- `value`: The new value.

tabBarLocation ()
Where to place the tab bar.

Either `'top'`, `'left'`, `'bottom'` or `'right'`.

setTabBarLocation (value)
Setter for *tabBarLocation()*.

Parameters
- `value`: The new value.

switchInvisibleAnimationName ()
Optional animation name for switching away from a tab.

setSwitchInvisibleAnimationName (value)
Setter for *switchInvisibleAnimationName()*.

Parameters
- `value`: The new value.

switchVisibleAnimationName ()
Optional animation name for switching to a tab.

setSwitchVisibleAnimationName (value)
Setter for *switchVisibleAnimationName()*.

Parameters
- `value`: The new value.

switchInvisibleAnimationDuration ()
Optional animation duration (in msec) for the switch away animation.

See also *clove::TabView::switchInvisibleAnimationName()*.

setSwitchInvisibleAnimationDuration(value)
Setter for *switchInvisibleAnimationDuration()*.

Parameters
- `value`: The new value.

switchVisibleAnimationDuration()
Optional animation duration (in msec) for the switch animation.

See also *clove::TabView::switchVisibleAnimationName()*.

setSwitchVisibleAnimationDuration(value)
Setter for *switchVisibleAnimationDuration()*.

Parameters
- `value`: The new value.

OnTabCreationRequested
Triggered when the user requested a new tab.

See also *userMayAddTabs()*.

This is a *clove.Event* instance. See Manual for details.

addTab(config)
Adds a new tab.

The configuration may contain the following additional keys:
- `tabLabel`: The tab header text.
- `tabIcon`: The tab icon as *clove::Icon*.
- `tabMayBeClosedByUser`: If the user may close this tab.

This method returns a *clove::Widget* which can be used for getting subwidgets or removing the tab.
Parameters
- `config`: A widget configuration like for *clove::build()*.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

setProperty(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`getPreferredHeightForWidth``(w)`

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass``(clss)`

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass``(clss)`

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.
Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class CheckButton` : **`public`** *clove*::*Widget*

A check button.

The user can choose to check and uncheck freely (in contrast to *clove::RadioButton*).

Public Functions

`label`()

The label text.

`setLabel`(value)

Setter for *label()*.

Parameters

- `value`: The new value.

`checked`()

If this check button is checked.

`setChecked`(value)

Setter for *checked()*.

Parameters

- `value`: The new value.

`OnChanged`

Triggered when this check button was selected.

This is a *clove.Event* instance. See Manual for details.

`OnClicked`

Triggered when this check button was selected. Same as OnChanged.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be ‘undefined’ for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()

Sets the focus to this widget.

`isAlive`()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth`()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

setStyleClass (value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style ()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle (value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility ()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility (value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing ()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing (value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus ()
If the widget can have the keyboard focus.

setMayFocus (value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity ()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity (value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity ()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity (value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus()`

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize()`

Returns inner width and height of this widget.

`outerSize()`

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class DataBinding`

A data binding.

Read the Manual for details.

Public Functions

DataBinding(widget, propertyName, datasource, ptr, dataDirection, valueComparator, valueConverter)

Parameters
- `widget`: The *clove::Widget* to bind to the datasource.
- `propertyName`: The property name in the widget to bind.
- `datasource`: The *clove::Datasource* to bind to the widget.
- `ptr`: The *clove::DatasourceValuePointer*.
- `dataDirection`: See *clove::databind()*.
- `valueComparator`: Optional custom function which checks if two values are equal.
- `valueConverter`: Optional *clove::DataBindingConverter* for conversion between ui and data model.

bind()

Activates the binding.

unbind()

Deactivates the binding.

class DataBindingConverter

A data binding converter maps values between ui representation and data model in a custom way.

Public Functions

convertTo(x)

Converts an original datasource value to a user interface representation.

Parameters
- `x`: The original datasource value.

convertFrom(y)

Converts a user interface representation to an original datasource value.

Parameters
- `y`: The user interface representation value.

class Datasource

Base class for datasources.

Read the Manual for details.

Subclassed by *clove::AsyncDatasource*, *clove::NativeDatasource*, *clove::ProxyDatasource*

Public Functions

getValue(ptr)

Returns the value for a given node.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata(ptr)

Returns an object of metadata information (like icons, status infos used for styling, …). Optional.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

`changeValue`(ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

`rowCount`(*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`columnCount`(*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`valuePointer`(irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent`(ptr)

Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

`valuePointerNavigateInDepth`(ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

`OnDataInsert`

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`

Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`class DatasourceValuePointer`

A pointer to one node in a *clove::Datasource*.

Public Functions

DatasourceValuePointer(irow, icol, backendObject, datasource)

Typically called only inside a *clove::Datasource* implementation. From outside, you should use *clove::Datasource::valuePointer*.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `backendObject`: The backend object. This depends on the datasource implementation.
- `datasource`: The owning *clove::Datasource*.

irow

The row index.

icol

The column index.

datasource

The owning *clove::Datasource*.

backendObject

The backend object.

This depends on the datasource implementation.

parent()

Returns the parent *clove::DatasourceValuePointer*.

sibling(irow, icol)

Returns a sibling *clove::DatasourceValuePointer*.

Parameters

- `irow`: The row index.
- `icol`: The column index.

rowCount()

Returns the number of rows in this node.

columnCount()

Returns the number of columns in this node.

getValue()

Returns the value stored behind this pointer.

static **equals**(ptr1, ptr2)

Checks if two value pointers point to the same place.

Parameters

- `ptr1`: The first *clove::DatasourceValuePointer*.
- `ptr2`: The second *clove::DatasourceValuePointer*.

static **toPath**(ptr)

Returns an array-of-tuples representation for a value pointer.

See also *clove::DatasourceValuePointer::fromPath()*.

Parameters

- `ptr`: The *clove::DatasourceValuePointer* to convert.

static **fromPath**(datasource, path)

Returns a *clove::DatasourceValuePointer* for a array-of-tuples representation.

See also *clove::DatasourceValuePointer::toPath()*.

Parameters
- `datasource`: The *clove::Datasource* the value pointer shall correspond to.
- `path`: The array-of-tuples.

static toString(ptr)
Returns a string representation for a value pointer.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

class DataView : **public** *clove*::*Widget*
Base class for some views which shows hierarchical data from a *clove::Datasource*.

Read the Manual for details.

Although this is a fully-functional and implemented class, you should consider using one of the subclasses.

Subclassed by *clove::ListView*, *clove::TableView*, *clove::TreeView*

Public Functions

datasource()
The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

setDatasource(value)
Setter for *datasource()*.

Parameters
- `value`: The new value.

dataViewProcessor()
An optional `function(ptr, value)` for processing a datasource value to a display string.

setDataViewProcessor(value)
Setter for *dataViewProcessor()*.

Parameters
- `value`: The new value.

cellGenerator()
A generator function for complex cell content.

This method is called for each cell with `(clove::Widget, clove::DatasourceValuePointer)`. It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

setCellGenerator(value)
Setter for *cellGenerator()*.

Parameters
- `value`: The new value.

alwaysAllocateExpanderSpace()
If some space is allocated for an expander even for cells without children.

setAlwaysAllocateExpanderSpace(value)
Setter for *alwaysAllocateExpanderSpace()*.

Parameters
- `value`: The new value.

`showOnlyFirstColumn``()`
If to show only the first column and hide the other ones.

`setShowOnlyFirstColumn``(value)`
Setter for *showOnlyFirstColumn()*.

Parameters
- `value`: The new value.

`hideExpanders``()`
If to hide all expanders.

`setHideExpanders``(value)`
Setter for *hideExpanders()*.

Parameters
- `value`: The new value.

`allowSelection``()`
If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

`setAllowSelection``(value)`
Setter for *allowSelection()*.

Parameters
- `value`: The new value.

`allowChecking``()`
If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

`setAllowChecking``(value)`
Setter for *allowChecking()*.

Parameters
- `value`: The new value.

`granularity``()`
The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

`setGranularity``(value)`
Setter for *granularity()*.

Parameters
- `value`: The new value.

`showChangeMenu``()`
If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:
- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

`setShowChangeMenu`(value)
Setter for *showChangeMenu()*.

Parameters
- `value`: The new value.

`editOnGesture`()
If the user shall be able to edit cells by double clicking/tapping them.

`setEditOnGesture`(value)
Setter for *editOnGesture()*.

Parameters
- `value`: The new value.

`rowsResizable`()
If the user shall be able to resize rows.

`setRowsResizable`(value)
Setter for *rowsResizable()*.

Parameters
- `value`: The new value.

`columnsResizable`()
If the user shall be able to resize columns.

`setColumnsResizable`(value)
Setter for *columnsResizable()*.

Parameters
- `value`: The new value.

`selectCell`(ptr)
Selects a cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

`isCellSelected`(ptr)
Checks if a given cell is selected.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

`checkedCells`()
Returns the checked cells as list of *clove::DatasourceValuePointer*.

`setCheckedCells`(ptrs)
Seturns the checked cells.

Parameters
- `ptrs`: A list of *clove::DatasourceValuePointer*.

`isCellChecked`(ptr)
Checks if a given cell is checked.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

`setCellChecked`(ptr, val)
Sets if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

selection()
Returns the selected item as *clove::DatasourceValuePointer*.

headersource()
The *clove::Headersource* providing row and column header configurations.

setHeadersource(value)
Setter for *headersource()*.

Parameters
- `value`: The new value.

gridVisible()
If to show a cell grid.

setGridVisible(value)
Setter for *gridVisible()*.

Parameters
- `value`: The new value.

nodeActivationNeedsDoubleClick()
If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

setNodeActivationNeedsDoubleClick(value)
Setter for *nodeActivationNeedsDoubleClick()*.

Parameters
- `value`: The new value.

expandCell(ptr)
Expands a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

expandCellRecursive(ptr)
Expands a given cell and all parents.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

collapseCell(ptr)
Collapses a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

isCellExpanded(ptr)
Checks if a given cell is expanded.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

editCell(ptr)
Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.
Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`OnSelectionChanged`
Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

`declareProperty` (k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty` (k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

`setProperty` (k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

`bindProperty` (k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init` (rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.
Parameters
- `rootNameScope`: The root namescope to add this widget to.

`doinit` ()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

`getPreferredHeightForWidth``(w)`

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope``()`

The namescope this widget contains to.

`nameScope``()`

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled``()`
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility``()`
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets``()`
List of the children *clove::Widget* instances.

`parentWidget``()`
The parent *clove::Widget*.

`name``()`
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName``(value)`
Setter for *name()*.

Parameters
- `value`: The new value.

`enabled``()`
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled``(value)`
Setter for *enabled()*.

Parameters
- `value`: The new value.

`styleClass``()`
Custom css class(es).

`setStyleClass``(value)`
Setter for *styleClass()*.

Parameters
- `value`: The new value.

`style``()`
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(value)`
Setter for *style()*.

Parameters
- `value`: The new value.

`visibility``()`
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class DateBox` : **`public`** *clove*::*EditBox*
A user input box for date input.

The actual behavior depends on the browser.

Public Functions

`date`()
The date value in the box as JavaScript Date.

`setDate`(value)
Setter for *date()*.

Parameters
- `value`: The new value.

`readOnly`()
If the edit box is read-only or editable by the user.

`setReadOnly`(value)
Setter for *readOnly()*.

Parameters
- `value`: The new value.

`text`()
The current text.

`setText`(value)
Setter for *text()*.

Parameters

- `value`: The new value.

`hintText`()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems`()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Dialog` : **`public`** *clove*::*Widget*

A dialog is a container for a new sub user interface, decorated with a border and a title bar.

Public Functions

`body`()

The inner widget as widget configuration like for *clove::build()*.

`setBody`(value)

Setter for *body()*.

Parameters

- `value`: The new value.

`title`()

The dialog title text.

`setTitle`(value)

Setter for *title()*.

Parameters

- `value`: The new value.

`titleicon`()

The dialog title icon as *clove::Icon*.

`setTitleicon`(value)

Setter for *titleicon()*.

Parameters

- `value`: The new value.

`close`()

Closes and removes this dialog.

`static show`(dlg, showconfig)

Shows a dialog.

The root view of `dlg` is expected to be a *clove::Dialog* (or a subclass).

Parameters

- `dlg`: The widget configuration, as for *clove::build()*, which specifies the dialog.
- `showconfig`: A configuration object which specifies some presentation aspects.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class DropdownBox` : **`public`** *clove*::*EditComboBox*

A single-line text box (without edit functionality) with a popup list of values to select from.

Public Functions

`popupItems``()`

A *clove.Datasource* with a list of items for the popup.

`setPopupItems``(value)`

Setter for *popupItems()*.

Parameters

- `value`: The new value.

`readOnly``()`

If the edit box is read-only or editable by the user.

`setReadOnly``(value)`

Setter for *readOnly()*.

Parameters

- `value`: The new value.

`text``()`

The current text.

`setText``(value)`

Setter for *text()*.

Parameters

- `value`: The new value.

`hintText`()
The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(value)
Setter for *hintText()*.

Parameters
- `value`: The new value.

`autocompletionItems`()
A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(value)
Setter for *autocompletionItems()*.

Parameters
- `value`: The new value.

`autocompletionFilter`()
How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)
Setter for *autocompletionFilter()*.

Parameters
- `value`: The new value.

`autocompletionOpenForNoText`()
If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)
Setter for *autocompletionOpenForNoText()*.

Parameters
- `value`: The new value.

`setTextSelection`(ifrom, ito)
Selects the specified part of the text.

Parameters
- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()
Returns the current text selection indices.

`OnChanged`
Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`
Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class` **`EditBox`** : `public` *clove*::*Widget*

A single-line box for text input from the user.

Subclassed by *clove::DateBox*, *clove::EditComboBox*, *clove::MultilineEditBox*, *clove::NumericEditBox*, *clove::PasswordEditBox*, *clove::TimeBox*

Public Functions

`readOnly``()`

If the edit box is read-only or editable by the user.

`setReadOnly``(value)`

Setter for *readOnly()*.

Parameters

- `value`: The new value.

`text``()`

The current text.

`setText``(value)`

Setter for *text()*.

Parameters

- `value`: The new value.

`hintText``()`

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText``(value)`

Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems``()`

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

setAutocompletionItems(value)
Setter for *autocompletionItems()*.

Parameters
- `value`: The new value.

autocompletionFilter()
How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

setAutocompletionFilter(value)
Setter for *autocompletionFilter()*.

Parameters
- `value`: The new value.

autocompletionOpenForNoText()
If to show the autocompletion list when the edit box is empty.

setAutocompletionOpenForNoText(value)
Setter for *autocompletionOpenForNoText()*.

Parameters
- `value`: The new value.

setTextSelection(ifrom, ito)
Selects the specified part of the text.

Parameters
- `ifrom`: The selection begin index.
- `ito`: The selection end index.

textSelection()
Returns the current text selection indices.

OnChanged
Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

OnPopupTextSelected
Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled ()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass ()
Custom css class(es).

setStyleClass (value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style ()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle (value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility ()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility (value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing ()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing (value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus ()
If the widget can have the keyboard focus.

setMayFocus (value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class EditComboBox` : **`public`** *clove*::*EditBox*

A single-line box for text input from the user with an additional popup list of value recommendations.

Subclassed by *clove::DropdownBox*

Public Functions

`popupItems``()`

A *clove.Datasource* with a list of items for the popup.

`setPopupItems``(value)`

Setter for *popupItems()*.

Parameters

- `value`: The new value.

`readOnly``()`

If the edit box is read-only or editable by the user.

`setReadOnly``(value)`

Setter for *readOnly()*.

Parameters

- `value`: The new value.

`text``()`

The current text.

`setText``(value)`

Setter for *text()*.

Parameters

- `value`: The new value.

`hintText``()`

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText``(value)`

Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems``()`

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems``(value)`

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

autocompletionFilter()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

setAutocompletionFilter(value)

Setter for *autocompletionFilter()*.

Parameters
- `value`: The new value.

autocompletionOpenForNoText()

If to show the autocompletion list when the edit box is empty.

setAutocompletionOpenForNoText(value)

Setter for *autocompletionOpenForNoText()*.

Parameters
- `value`: The new value.

setTextSelection(ifrom, ito)

Selects the specified part of the text.

Parameters
- `ifrom`: The selection begin index.
- `ito`: The selection end index.

textSelection()

Returns the current text selection indices.

OnChanged

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

OnPopupTextSelected

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters
- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()
Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass (clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass (clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()
The namescope this widget contains to.

nameScope ()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.
Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()
List of the children *clove::Widget* instances.

parentWidget ()
The parent *clove::Widget*.

name ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled ()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

class Event

An event. It can have some handlers (or: listeners) and can be triggered.

Read the Manual for details about the Clove event mechanism.

Public Functions

Event()

trigger(params)

Triggers the event.

This calls all registered handlers.

Parameters

- `params`: Additional information about the event incidence.

addHandler(fct, owner)

Adds a handler (or: listener) to this event.

Whenever the event is triggered afterwards, the new handler is called.

Parameters

- `fct`: A handler function.
- `owner`: Optional owner widget. If given, removeHandler will be called automatically after widget removal.

hasHandlers()

Checks if this event has any handlers (otherwise triggering it doesn't have an effect).

addHandlerOnce(fct, owner)

Like *addHandler()*, but automatically removes the handler after the event occurred once.

Parameters

- `fct`: A handler function.
- `owner`: Optional owner widget. If given, removeHandler will be called automatically after widget removal.

removeHandler(fct)

Removes a handler from this event.

Parameters

- `fct`: The handler function to remove.

class EventArgs

Arguments for a particular incidence of a *clove::Event*.

It carries some additional informations about it. The details depend on the event (and the component which triggers it).

Public Functions

EventArgs(params)

Parameters

- `params`: Additional information about the event incidence.

skipExecution()

Marks the further event handling for skipping.

class Expander : **public** *clove*::*Widget*

A single-widget container which can be expanded or collapsed.

Public Functions

`isExpanded`()

If the expander is expanded.

`setIsExpanded`(value)

Setter for *isExpanded()*.

Parameters
- `value`: The new value.

`collapsedIcon`()

The icon for the expander when collapsed.

`setCollapsedIcon`(value)

Setter for *collapsedIcon()*.

Parameters
- `value`: The new value.

`collapsedLabel`()

The label for the expander when collapsed.

`setCollapsedLabel`(value)

Setter for *collapsedLabel()*.

Parameters
- `value`: The new value.

`expandedIcon`()

The icon for the expander when expanded.

`setExpandedIcon`(value)

Setter for *expandedIcon()*.

Parameters
- `value`: The new value.

`expandedLabel`()

The label for the expander when expanded.

`setExpandedLabel`(value)

Setter for *expandedLabel()*.

Parameters
- `value`: The new value.

`body`()

The inner *clove::Widget*.

`setBody`(value)

Setter for *body()*.

Parameters
- `value`: The new value.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.

- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class FilterProxyDatasource` : **`public`** *clove*::*ProxyDatasource*

A *clove::Datasource* implementation which filters another *clove::Datasource*.

Public Functions

`FilterProxyDatasource`(datasource, rowfilterfct, colfilterfct)

Parameters

- `datasource`: The source *clove::Datasource*.
- `rowfilterfct`: A `function(datasource, idx, parent)` returning `true` if the filter accepts this row (optional).
- `colfilterfct`: A `function(datasource, idx, parent)` returning `true` if the filter accepts this column (optional).

`setRowFilter`(rowfilterfct)

Sets the row filter function.

Parameters

- `rowfilterfct`: The row filter function. See constructor for details.

`setColumnFilter`(colfilterfct)

Sets the column filter function.

Parameters

- `colfilterfct`: The column filter function. See constructor for details.

`setDatasource`(datasource)

Sets the source datasource.

Parameters

- `datasource`: The source *clove::Datasource*.

`proxyPointerToNativePointer`(proxyptr)

Translates a *clove::DatasourceValuePointer* from this proxy datasource to a one for the source datasource.

Parameters

- `proxyptr`: A value pointer.

`nativePointerToProxyPointer`(nativeptr)

Translates a *clove::DatasourceValuePointer* from the source datasource to a one for this proxy datasource.

Parameters

- `nativeptr`: A value pointer.

`refresh`()

Refreshes the filtering.

Call this when the outer conditions changed and the filters must be applied again.

getValue (ptr)

Returns the value for a given node.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata (ptr)

Returns an object of metadata information (like icons, status infos used for styling, . . .). Optional.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

changeValue (ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount (*parent*)

Returns the number of rows for a given node.

Parameters
- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount (*parent*)

Returns the number of columns for a given node.

Parameters
- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters
- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

parent (ptr)

Returns the parent node for a given node.

Parameters
- `ptr`: A node as *clove::DatasourceValuePointer*.

valuePointerNavigateInDepth (ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters
- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node’s children are to be traversed.

OnDataInsert

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`

Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)

Returns the header configuration for a given row.

Parameters
- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader` (irow, *parent*)

Returns the header configuration for a given column.

Parameters
- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible` (*parent*)

If row headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible` (*parent*)

If column headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

`class FlatLayout` : **`public`** *clove*::*Layout*

A mixin for flat layout implementations.

This layout shows one of its childs in full sizes and hides all the others.

Subclassed by *clove::FlatView*

Public Functions

`switchInvisibleAnimationName`()
Optional animation name for disabling a view.

`setSwitchInvisibleAnimationName`(value)
Setter for *switchInvisibleAnimationName()*.

Parameters
- `value`: The new value.

`switchVisibleAnimationName`()
Optional animation name for enabling a view.

`setSwitchVisibleAnimationName`(value)
Setter for *switchVisibleAnimationName()*.

Parameters
- `value`: The new value.

`switchInvisibleAnimationDuration`()
Optional animation duration (in msec) for the disabling animation.

See also *clove::FlatLayout::switchInvisibleAnimationName()*.

`setSwitchInvisibleAnimationDuration`(value)
Setter for *switchInvisibleAnimationDuration()*.

Parameters
- `value`: The new value.

`switchVisibleAnimationDuration`()
Optional animation duration (in msec) for the enabling animation.

See also *clove::FlatLayout::switchVisibleAnimationName()*.

`setSwitchVisibleAnimationDuration`(value)
Setter for *switchVisibleAnimationDuration()*.

Parameters
- `value`: The new value.

`collapseHidden`()
If the non-visible widgets shall be collapsed (affects layouting).

`setCollapseHidden`(value)
Setter for *collapseHidden()*.

Parameters
- `value`: The new value.

`currentView`()
The index (integer) of the currently visible child.

`setCurrentView`(value)
Setter for *currentView()*.

Parameters
- `value`: The new value.

`children`()
List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)
Setter for *children()*.

Parameters
- `value`: The new value.

`addChild`(value)
Adds a new child widget to the layout.

Parameters
- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds`()
Removes all childs from the layout.

`class FlatView` : **`public`** *clove*::*Widget*, **`public`** *clove*::*FlatLayout*
A container which shows one of its child widgets in full size and hides all others.

Public Functions

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

`setProperty`(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit``()`

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly``()`

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters
- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize``()`
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`switchInvisibleAnimationName``()`
Optional animation name for disabling a view.

`setSwitchInvisibleAnimationName``(value)`
Setter for *switchInvisibleAnimationName()*.

Parameters
- `value`: The new value.

`switchVisibleAnimationName``()`
Optional animation name for enabling a view.

`setSwitchVisibleAnimationName``(value)`
Setter for *switchVisibleAnimationName()*.

Parameters
- `value`: The new value.

`switchInvisibleAnimationDuration``()`
Optional animation duration (in msec) for the disabling animation.

See also *clove::FlatLayout::switchInvisibleAnimationName()*.

setSwitchInvisibleAnimationDuration(value)
Setter for *switchInvisibleAnimationDuration()*.

Parameters
- `value`: The new value.

switchVisibleAnimationDuration()
Optional animation duration (in msec) for the enabling animation.

See also *clove::FlatLayout::switchVisibleAnimationName()*.

setSwitchVisibleAnimationDuration(value)
Setter for *switchVisibleAnimationDuration()*.

Parameters
- `value`: The new value.

collapseHidden()
If the non-visible widgets shall be collapsed (affects layouting).

setCollapseHidden(value)
Setter for *collapseHidden()*.

Parameters
- `value`: The new value.

currentView()
The index (integer) of the currently visible child.

setCurrentView(value)
Setter for *currentView()*.

Parameters
- `value`: The new value.

children()
List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)
Setter for *children()*.

Parameters
- `value`: The new value.

addChild(value)
Adds a new child widget to the layout.

Parameters
- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()
Removes all childs from the layout.

class Form : **public** *clove*::*Widget*
A container for a form-like with a label for each entry in a row-oriented alignment.

Public Functions

sections()

A list of widget configurations as for *clove::build()*. Each defines one section in the form.

Define the labels in the `formSectionLabel` property of each widget.

setSections(value)

Setter for *sections()*.

Parameters
- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters
- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters
- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters
- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()

List of the children *clove::Widget* instances.

parentWidget ()

The parent *clove::Widget*.

name ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled ()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass ()

Custom css class(es).

setStyleClass (value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style ()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle (value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Grid` : **`public`** *clove*::*Widget*, **`public`** *clove*::*GridLayout*

A container for aligning child widgets in a grid.

Set the `row` and `col` property in the child widget configurations to some numbers for assigning them to cells.

Public Functions

`declareProperty``(`k, defaultV`)`

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty``(k)`

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters
- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters
- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout`()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus`()`

Sets the focus to this widget.

isAlive`()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth`()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth`()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth`(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth`(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth`()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth`()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth`(w)`

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth`(w)`

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

`removeStyleClass`(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

`isStyleClass`(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()
The namescope this widget contains to.

`nameScope`()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.
Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()
List of the children *clove::Widget* instances.

`parentWidget`()
The parent *clove::Widget*.

`name`()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)
Setter for *name()*.

Parameters
- `value`: The new value.

`enabled`()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

`styleClass`()
Custom css class(es).

`setStyleClass`(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

`style`()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)
Setter for *style()*.

Parameters
- `value`: The new value.

`visibility`()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`insertRow`(i)

Inserts a new empty row to the grid.

Parameters
- `i`: The row index.

`insertColumn`(i)

Inserts a new empty column to the grid.

Parameters
- `i`: The column index.

`removeRow`(i)

Removes a row from the grid.

Parameters
- `i`: The row index.

`removeColumn`(i)

Removes a column from the grid.

Parameters
- `i`: The column index.

`children`()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)

Setter for *children()*.

Parameters
- `value`: The new value.

`addChild`(value)

Adds a new child widget to the layout.

Parameters
- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds`()

Removes all childs from the layout.

`class GridLayout` : **`public`** *clove*::*Layout*

A mixin for grid layout implementations.

Subclassed by *clove::Grid*, *clove::StackLayout*

Public Functions

`insertRow`(i)

Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

`insertColumn`(i)

Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

`removeRow`(i)

Removes a row from the grid.

Parameters

- `i`: The row index.

`removeColumn`(i)

Removes a column from the grid.

Parameters

- `i`: The column index.

`children`()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)

Setter for *children()*.

Parameters

- `value`: The new value.

`addChild`(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds`()

Removes all childs from the layout.

`class Headersource`

Base class for headersources.

A headersource provides information for headers in data views. Read the Manual for details.

Subclassed by *clove::AjaxAsyncDatasource*, *clove::AsyncDatasource*, *clove::NativeDatasource*, *clove::ProxyDatasource*

Public Functions

getRowHeader (irow, parent)
Returns the header configuration for a given row.

Parameters
- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

getColumnHeader (irow, parent)
Returns the header configuration for a given column.

Parameters
- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

rowHeadersVisible (parent)
If row headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

columnHeadersVisible (parent)
If column headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

OnHeaderDataInsert
Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataRemove
Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataUpdate
Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

OnHeaderVisibilityUpdated
Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

class HorizontalStack : **public** *clove*::*Widget*, **public** *clove*::*StackLayout*
A container which stacks child widgets column-wise.

Public Functions

cols ()
The list of child widgets as widget configurations like for *clove::build()*.

setCols (value)
Setter for *cols()*.

Parameters
- `value`: The new value.

`declareProperty` (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty` (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty` (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty` (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init` (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit` ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly` ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility``()`
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets``()`
List of the children *clove::Widget* instances.

`parentWidget``()`
The parent *clove::Widget*.

`name``()`
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName``(value)`
Setter for *name()*.

Parameters
- `value`: The new value.

`enabled``()`
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled``(value)`
Setter for *enabled()*.

Parameters
- `value`: The new value.

`styleClass``()`
Custom css class(es).

`setStyleClass``(value)`
Setter for *styleClass()*.

Parameters
- `value`: The new value.

`style``()`
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(value)`
Setter for *style()*.

Parameters
- `value`: The new value.

`visibility``()`
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility``(value)`
Setter for *visibility()*.

Parameters
- `value`: The new value.

`doStandaloneResizing``()`
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`insertRow`(i)

Inserts a new empty row to the grid.

Parameters
- `i`: The row index.

`insertColumn`(i)

Inserts a new empty column to the grid.

Parameters
- `i`: The column index.

`removeRow`(i)

Removes a row from the grid.

Parameters
- `i`: The row index.

`removeColumn`(i)

Removes a column from the grid.

Parameters
- `i`: The column index.

`children`()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)

Setter for *children()*.

Parameters
- `value`: The new value.

`addChild`(value)

Adds a new child widget to the layout.

Parameters
- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds`()

Removes all childs from the layout.

class HtmlView : **public** *clove*::*Widget*

A host for arbitrary html content.

Public Functions

contentRoot()

The root dom node of the html content.

setContentRoot(value)

Setter for *contentRoot()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

`visibility``()`

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility``(value)`

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing``()`

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing``(value)`

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus``()`

If the widget can have the keyboard focus.

`setMayFocus``(value)`

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity``()`

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity``(value)`

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity``()`

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity``(value)`

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing``()`

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing``(value)`

Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *verticalStretchAffinity()*==0.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class I18N`
Internationalization support class.

Always use the instance *clove.i18n*. Read the Manual for details.

Public Functions

`addString` (id, texts)
Adds a text to the internationalization text table.

The keys are ISO 639-1 language codes.

Example: \`addString('PleaseWait', {'de':'Bitte warten', 'en':'Please wait'})\`
Parameters
- `id`: The text identifier name, like 'FooBar'.
- `texts`: A configuration object which holds a text representation for all target languages.

`setLanguage` (langcode)
Sets the current language.

You don't need to call this function, unless you want to override the user's language as it is configured in the operating system or browser.
Parameters

- `langcode`: The new current language (as ISO 639-1 language code).

static parseLangCode (lang)

Returns a clove string representation.

Only indended to be used by the infrastructure.

Parameters

- `lang`: A language code string as understood by *addString()*.

class Icon

An icon.

Can be shown at many places, like in buttons, menus or *clove::IconView*.

Construct instances with *clove::Icon::byUrl()* or *clove::Icon::bySymbol()*.

Public Functions

static byUrl (url)

Creates a *clove::Icon* by a url pointing to an image.

Parameters

- `url`: The image url.

static bySymbol (chr)

Creates a *clove::Icon* by a text character.

Parameters

- `chr`: The text character.

createHtml (height)

Creates an html representation.

This method is used by widget implementations.

Parameters

- `height`: The icon height in pixels.

class IconView : public *clove*::*Widget*

Shows an icon.

The icon can come from an image file or from a Unicode symbol.

Public Functions

size ()

The icon size as css length.

setSize (value)

Setter for *size()*.

Parameters

- `value`: The new value.

icon ()

The *clove::Icon* to show.

setIcon (value)

Setter for *icon()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class ImageView` : **`public`** *clove*::*Widget*
Shows an image.

This does not contain any viewer controls like zoom buttons.

Public Functions

`source``()`
The image source URL.

`setSource``(value)`
Setter for *source()*.

Parameters
- `value`: The new value.

`zoom``()`
The zoom level.

1.0 is normal, larger values zoom in, smaller values zoom out. Specify 'auto' for automatically adjusting it to the available room.

`setZoom``(value)`
Setter for *zoom()*.

Parameters
- `value`: The new value.

`keepAspectRatio``()`
For 'auto' zoom: Whether to keep aspect ratio of the image.

`setKeepAspectRatio``(value)`
Setter for *keepAspectRatio()*.

Parameters
- `value`: The new value.

`OnLoaded`
Triggered when the image loading ended.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Label` : **`public`** *clove*::*Widget*

A text label.

Public Functions

`label`()

The label text.

For rich content, use *htmlContent()* instead.

`setLabel`(value)

Setter for *label()*.

Parameters

- `value`: The new value.

`htmlContent`()

The label content as html.

For plain text, use *label()* instead.

`setHtmlContent`(value)

Setter for *htmlContent()*.

Parameters

- `value`: The new value.

`focusBuddy`()

A *clove::Widget* or a widget name for a focus buddy.

Whenever the user clicks on this label, it will focus the buddy.

`setFocusBuddy`(value)

Setter for *focusBuddy()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Layout`

A mixin for a widget classes which align child widgets in some way.

This is a base type and has some more interesting sub-classes.

Subclassed by *clove::FlatLayout*, *clove::GridLayout*, *clove::WrapLayout*

Public Functions

`children``()`

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren``(value)`

Setter for *children()*.

Parameters

- `value`: The new value.

`addChild``(value)`

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds``()`

Removes all childs from the layout.

`class ListView` : **`public`** *clove*::*DataView*

A view which shows a *clove::Datasource* in a list.

Public Functions

datasource()

The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

setDatasource(value)

Setter for *datasource()*.

Parameters

- `value`: The new value.

dataViewProcessor()

An optional `function(ptr, value)` for processing a datasource value to a display string.

setDataViewProcessor(value)

Setter for *dataViewProcessor()*.

Parameters

- `value`: The new value.

cellGenerator()

A generator function for complex cell content.

This method is called for each cell with (*clove::Widget*, *clove::DatasourceValuePointer*). It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

setCellGenerator(value)

Setter for *cellGenerator()*.

Parameters

- `value`: The new value.

alwaysAllocateExpanderSpace()

If some space is allocated for an expander even for cells without children.

setAlwaysAllocateExpanderSpace(value)

Setter for *alwaysAllocateExpanderSpace()*.

Parameters

- `value`: The new value.

showOnlyFirstColumn()

If to show only the first column and hide the other ones.

setShowOnlyFirstColumn(value)

Setter for *showOnlyFirstColumn()*.

Parameters

- `value`: The new value.

hideExpanders()

If to hide all expanders.

setHideExpanders(value)

Setter for *hideExpanders()*.

Parameters

- `value`: The new value.

`allowSelection`()

If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

`setAllowSelection`(value)

Setter for *allowSelection()*.

Parameters

- `value`: The new value.

`allowChecking`()

If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

`setAllowChecking`(value)

Setter for *allowChecking()*.

Parameters

- `value`: The new value.

`granularity`()

The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

`setGranularity`(value)

Setter for *granularity()*.

Parameters

- `value`: The new value.

`showChangeMenu`()

If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:

- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

`setShowChangeMenu`(value)

Setter for *showChangeMenu()*.

Parameters

- `value`: The new value.

`editOnGesture`()

If the user shall be able to edit cells by double clicking/tapping them.

`setEditOnGesture`(value)

Setter for *editOnGesture()*.

Parameters

- `value`: The new value.

`rowsResizable`()

If the user shall be able to resize rows.

setRowsResizable(value)
Setter for *rowsResizable()*.

Parameters
- `value`: The new value.

columnsResizable()
If the user shall be able to resize columns.

setColumnsResizable(value)
Setter for *columnsResizable()*.

Parameters
- `value`: The new value.

selectCell(ptr)
Selects a cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

isCellSelected(ptr)
Checks if a given cell is selected.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

checkedCells()
Returns the checked cells as list of *clove::DatasourceValuePointer*.

setCheckedCells(ptrs)
Seturns the checked cells.

Parameters
- `ptrs`: A list of *clove::DatasourceValuePointer*.

isCellChecked(ptr)
Checks if a given cell is checked.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

setCellChecked(ptr, val)
Sets if a given cell is checked.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

selection()
Returns the selected item as *clove::DatasourceValuePointer*.

headersource()
The *clove::Headersource* providing row and column header configurations.

setHeadersource(value)
Setter for *headersource()*.

Parameters
- `value`: The new value.

gridVisible()
If to show a cell grid.

setGridVisible(value)
Setter for *gridVisible()*.

Parameters
- `value`: The new value.

nodeActivationNeedsDoubleClick()
If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

setNodeActivationNeedsDoubleClick(value)
Setter for *nodeActivationNeedsDoubleClick()*.

Parameters
- `value`: The new value.

expandCell(ptr)
Expands a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

expandCellRecursive(ptr)
Expands a given cell and all parents.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

collapseCell(ptr)
Collapses a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

isCellExpanded(ptr)
Checks if a given cell is expanded.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

editCell(ptr)
Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.
Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

OnSelectionChanged
Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class MainView` : **`public`** *clove*::*Widget*
A typical main view, including a *clove::Toolbar* and a container for an inner body widget.

Public Functions

`sidebar`()
The sidebar widget as widget configuration like for *clove::build()*.

`setSidebar`(value)
Setter for *sidebar()*.

Parameters
- `value`: The new value.

`bodyLeft`()
The left inner widget as widget configuration like for *clove::build()*.

`setBodyLeft`(value)
Setter for *bodyLeft()*.

Parameters
- `value`: The new value.

`bodyRight`()
The right inner widget as widget configuration like for *clove::build()*.

`setBodyRight`(value)
Setter for *bodyRight()*.

Parameters
- `value`: The new value.

`bodyLeftViewActionLabel`()
The title string for the left inner widget.

`setBodyLeftViewActionLabel`(value)
Setter for *bodyLeftViewActionLabel()*.

Parameters
- `value`: The new value.

`bodyRightViewActionLabel`()
The title string for the right inner widget.

`setBodyRightViewActionLabel`(value)
Setter for *bodyRightViewActionLabel()*.

Parameters
- `value`: The new value.

splitterPosition()
The position of the splitter between the left and right inner widget as number between `0.0` and `1.0`.

setSplitterPosition(value)
Setter for *splitterPosition()*.

Parameters
- `value`: The new value.

switchToRightBody()
If in small mode, it switches the view to the right body.

switchToLeftBody()
If in small mode, it switches the view to the left body.

head1()
The 1st header text.

setHead1(value)
Setter for *head1()*.

Parameters
- `value`: The new value.

head2()
The 2nd header text.

setHead2(value)
Setter for *head2()*.

Parameters
- `value`: The new value.

headControl()
An optional widget for arbitrary control purposes as widget configuration like for *clove::build()*. It's displayed below the menu bar in full width.

setHeadControl(value)
Setter for *headControl()*.

Parameters
- `value`: The new value.

headControlWidget()
Returns the control widget as *clove::Widget* (or undefined if no *headControl()* is set).

icon()
The header icon as *clove::Icon*.

setIcon(value)
Setter for *icon()*.

Parameters
- `value`: The new value.

actions()
Menu actions.

See *clove::Menubar::actions()*.

setActions(value)
Setter for *actions()*.

Parameters

- `value`: The new value.

showOnly()
Specifies if to forcefully show just one of the two main panels (`0`, `1`, `undefined`).

setShowOnly(value)
Setter for *showOnly()*.

Parameters
- `value`: The new value.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

setProperty(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.
Parameters
- `rootNameScope`: The root namescope to add this widget to.

doinit()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class MediaPlayer` : **`public`** *clove*::*Widget*

A base class for media player widgets.

This is an abstract class. Use one of the subclasses.

Note: This interface provides actions and events for typical usage. For other scenarios, use *nativeNode()* for retrieving the `<audio>` or `<video>` html dom node. They provide a more powerful api.

Subclassed by *clove::AudioPlayer*, *clove::VideoPlayer*

Public Functions

`autoPlay``()`

If to automatically start playback as soon as possible.

`setAutoPlay``(value)`

Setter for *autoPlay()*.

Parameters

- `value`: The new value.

`showControls``()`

If to show user controls for play, pause and more.

`setShowControls``(value)`

Setter for *showControls()*.

Parameters

- `value`: The new value.

currentMediaPosition()
The current media playback position in seconds.

setCurrentMediaPosition(value)
Setter for *currentMediaPosition()*.

Parameters
- `value`: The new value.

loop()
If to playback in an endless loop.

setLoop(value)
Setter for *loop()*.

Parameters
- `value`: The new value.

muted()
If to mute the audio playback.

setMuted(value)
Setter for *muted()*.

Parameters
- `value`: The new value.

preload()
If to begin loading the media data as soon as possible.

setPreload(value)
Setter for *preload()*.

Parameters
- `value`: The new value.

volume()
The audio playback volume between `0.0` and `1.0`.

setVolume(value)
Setter for *volume()*.

Parameters
- `value`: The new value.

source()
The media source URL.

setSource(value)
Setter for *source()*.

Parameters
- `value`: The new value.

duration()
The duration of the loaded media source in seconds.

hasEnded()
If the playback has ended (i.e. reached the end).

isPaused()
If the playback is logically paused.

This does not return `true` just when loading stalls.

nativeNode()
Returns the native html dom node.

play()
Starts playback.

pause()
Pauses playback.

OnLoadingAborted
Triggered when the media loading aborted for some reasons (e.g. network errors).

This is a *clove.Event* instance. See Manual for details.

OnReadyForPlayback
Triggered when there are enough data for starting playback.

This is a *clove.Event* instance. See Manual for details.

OnDurationChanged
Triggered when the playback duration changed.

See also *duration()*.

This is a *clove.Event* instance. See Manual for details.

OnHasEnded
Triggered when the playback has ended.

See also *hasEnded()*.

This is a *clove.Event* instance. See Manual for details.

OnIsPaused
Triggered when the playback logically paused.

This is not triggered when loading stalls.

This is a *clove.Event* instance. See Manual for details.

OnIsPlaying
Triggered when the playback logically starts.

This is not triggered when loading has stalled and resumes.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

class Menubar : **public** *clove*::*AbstractMenu*

A menu bar.

Public Functions

actions()

The actions to be shown in this menu.

It is a list of action configuration objects, each having a structure like `{name:'myaction', label:'My menu action'}. `It can have a list of sub-items in the` subactions` property.

Instead of a simple list, it may also be a *clove::Datasource* with the action configuration structures aligned in rows.

Use *clove::MenuSeparator* for a separator.

One action allows this properties:
- `name`: The action name.
- `label`: The label string.
- `icon`: A *clove::Icon*.
- `disabled`: If it is disabled.
- `invisible`: If it is invisible.
- `checkable`: If it is checkable.
- `checked`: If it is checked.

setActions(value)

Setter for *actions()*.

Parameters
- `value`: The new value.

OnActionTriggered

Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

OnBeforeSubactionsExpanded

Triggered just before a branch of subaction is expanded.

Implement this event for populating it dynamically.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters
- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters
- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class ModalPanel` : **`public`** *clove*::*Widget*

A surface for implementing modality.

For a certain time, it will be inserted before the main user interface (i.e. higher on z-axis) in order to block user interaction with it.

It is typically not required to use it directly. Try *clove::Dialog::show()* and *clove::utils::popup()* before.

Public Functions

`fullyTransparent`()

If the modal panel is fully transparent (i.e. not really visible) instead of semi-transparent.

`setFullyTransparent`(value)

Setter for *fullyTransparent()*.

Parameters

- `value`: The new value.

`OnClicked`

Triggered when the user clicks on this modal panel.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass (clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()
The namescope this widget contains to.

nameScope ()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.
Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()
List of the children *clove::Widget* instances.

parentWidget ()
The parent *clove::Widget*.

name ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled ()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class MultilineEditBox` : **`public`** *clove*::*EditBox*

A multi-line box for text input from the user.

Public Functions

`readOnly`()
If the edit box is read-only or editable by the user.

`setReadOnly`(value)
Setter for *readOnly()*.

Parameters
- `value`: The new value.

`text`()
The current text.

`setText`(value)
Setter for *text()*.

Parameters
- `value`: The new value.

`hintText`()
The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(value)
Setter for *hintText()*.

Parameters
- `value`: The new value.

`autocompletionItems`()
A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(value)
Setter for *autocompletionItems()*.

Parameters
- `value`: The new value.

`autocompletionFilter`()
How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)
Setter for *autocompletionFilter()*.

Parameters
- `value`: The new value.

`autocompletionOpenForNoText`()
If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)
Setter for *autocompletionOpenForNoText()*.

Parameters
- `value`: The new value.

`setTextSelection`(ifrom, ito)
Selects the specified part of the text.

Parameters
- `ifrom`: The selection begin index.
- `ito`: The selection end index.

textSelection()

Returns the current text selection indices.

OnChanged

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

OnPopupTextSelected

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters
- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters
- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class` **`NameScope`**

A mixin realizing a new namescope.

Read the Manual for details.

Subclassed by *clove::RootNameScope*

Public Functions

`getByName`(name)

Finds a widget by name within this namescope.

Parameters

- `name`: The widget name.

`addChildNameScope`(childnamescope)

Adds a namescope to the children of this one, so a searcher will also search in this child.

Parameters

- `childnamescope`: The child namescope.

class NativeDatasource : **public** *clove*::*Datasource*, **public** *clove*::*Headersource*

A *clove::Datasource* implementation which stores all data in-memory.

It provides an interface for populating it with data and can directly be constructed and used. It also implements *clove::Headersource*, so it can also carry row and column header configurations.

Public Functions

NativeDatasource (config)

The configuration object allows the following keys:

- `readonly`: If this datasource may be changed from outside.
- `rowHeadersVisible`: If row headers shall be visible in presentation.
- `columnHeadersVisible`: If column headers shall be visible in presentation.

  Parameters
  - `config`: A configuration object.

root ()

The root node as clove::NativeDatasourceValue.

valuePointerToNativeNode (ptr)

Returns the clove::NativeDatasourceValue for a node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

insertRow (ir, *parent*)

Inserts a new empty row.

Parameters

- `ir`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

insertColumn (ic, *parent*)

Inserts a new empty column.

Parameters

- `ic`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

appendRow (*parent*)

Appends a new empty row (like inserting to the end).

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

appendColumn (*parent*)

Appends a new empty column (like inserting to the end).

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

removeRow (ir, *parent*)

Removes a row.

Parameters

- `ir`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

removeColumn (ic, *parent*)

Removes a column.

Parameters
- `ic`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

setValue (ptr, value)
Sets a new value to a node.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new node value.

setRootValue (value)
Sets a new value to the root node.

This is just a shorter variant of `setValue(undefined, value)` (for scalar usage).

Parameters
- `value`: The new node value.

setMetadata (ptr, key, value)
Sets a metadata key for a node.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.
- `key`: The metadata key, like 'icon'.
- `value`: The new value.

setRowHeader (irow, *parent*, headercfg)
Sets a row header configuration.

Parameters
- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.
- `headercfg`: The header configuration.

setColumnHeader (icol, *parent*, headercfg)
Sets a column header configuration.

Parameters
- `icol`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.
- `headercfg`: The header configuration.

getValue (ptr)
Returns the value for a given node.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata (ptr)
Returns an object of metadata information (like icons, status infos used for styling, . . .). Optional.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

changeValue (ptr, value)
Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

`rowCount` (*parent*)
Returns the number of rows for a given node.

Parameters
- `parent`: The parent as *clove::DatasourceValuePointer*.

`columnCount` (*parent*)
Returns the number of columns for a given node.

Parameters
- `parent`: The parent as *clove::DatasourceValuePointer*.

`valuePointer` (irow, icol, *parent*)
Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters
- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent` (ptr)
Returns the parent node for a given node.

Parameters
- `ptr`: A node as *clove::DatasourceValuePointer*.

`valuePointerNavigateInDepth` (ptr, direction, mayexpandfct)
Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters
- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

`OnDataInsert`
Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`
Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`
Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)
Returns the header configuration for a given row.

Parameters
- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader` (irow, *parent*)
Returns the header configuration for a given column.

Parameters
- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

rowHeadersVisible(*parent*)
If row headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

columnHeadersVisible(*parent*)
If column headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

OnHeaderDataInsert
Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataRemove
Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataUpdate
Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

OnHeaderVisibilityUpdated
Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

class NativeDatasourceNode
A node value implementation for *clove::NativeDatasource*.

Not intended for direct construction!

Public Functions

rowCount()
Returns the number of children rows.

columnCount()
Returns the number of children columns.

toRow(ir)
Returns the *clove::NativeDatasourceNode* which is in the same column as this one, but in a different row.

Parameters
- `ir`: The row index.

toColumn(ic)
Returns the *clove::NativeDatasourceNode* which is in the same row as this one, but in a different column.

Parameters
- `ic`: The column index.

valuePointer()
Returns a *clove::DatasourceValuePointer* pointing to this node.

getValue()
Returns the node value.

getMetadata()
Returns the node metadata.

See also *clove::Datasource::getMetadata()*.

setValue(v)
Sets the node value.

Parameters
- `v`: The new value.

setMetadata(k, v)
Sets a node metadata value.

Parameters
- `k`: The metadata key, like 'icon'.
- `v`: The new value.

removeRow(ir)
Removes a row in this node.

Parameters
- `ir`: The row index.

removeColumn(ic)
Removes a column in this node.

Parameters
- `ic`: The column index.

addRow(vals)
Adds a row to this node.

Parameters
- `vals`: A list of new values (one for each column).

addColumn(vals)
Adds a column to this node.

Parameters
- `vals`: A list of new values (one for each row).

insertRow(i, vals)
Inserts a row to this node.

Parameters
- `i`: The insertion position.
- `vals`: A list of new values (one for each column).

insertColumn(i, vals)
Inserts a column to this node.

Parameters
- `i`: The insertion position.
- `vals`: A list of new values (one for each row).

getChild(irow, icol)
Returns a child *clove::NativeDatasourceNode*.

Parameters
- `irow`: The row index.

- `icol`: The column index.

class NotificationController

Controller for clove notifications.

Notifications are small popups, typically in the top/right corner of the window. They can inform the user about some status changes or provide some ways for user interactions.

Use *clove::notifications*.

Public Functions

notify(config, notificationConfig)

Opens a new notification.

notificationConfig may contain:

- `timeout`: Closes the notification automatically after this amount of seconds.
- `closeable`: If to provide a close button.
- `priority`: Controls stuff like ordering. Default is 0, higher values mean more importance.

Return The *clove::Widget* implementing the notification. Use it for operations like removal or getting subwidget. This widget spans a new *clove::NameScope*!

Parameters

- `config`: A widget configuration for the notification body, as for *clove::build()*.
- `notificationConfig`: Some configuration aspects about the notification.

class NumericEditBox : **public** *clove*::*EditBox*

A text box with up/down buttons for numbers.

Public Functions

min()

The minimum value.

setMin(*value*)

Setter for *min()*.

Parameters

- `value`: The new value.

max()

The maximum value.

setMax(*value*)

Setter for *max()*.

Parameters

- `value`: The new value.

stepsize()

The step size.

setStepsize(*value*)

Setter for *stepsize()*.

Parameters

- `value`: The new value.

value()

The current numeric value.

`setValue`(*value*)
Setter for *value()*.

Parameters
- `value`: The new value.

`readOnly`()
If the edit box is read-only or editable by the user.

`setReadOnly`(*value*)
Setter for *readOnly()*.

Parameters
- `value`: The new value.

`text`()
The current text.

`setText`(*value*)
Setter for *text()*.

Parameters
- `value`: The new value.

`hintText`()
The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(*value*)
Setter for *hintText()*.

Parameters
- `value`: The new value.

`autocompletionItems`()
A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(*value*)
Setter for *autocompletionItems()*.

Parameters
- `value`: The new value.

`autocompletionFilter`()
How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(*value*)
Setter for *autocompletionFilter()*.

Parameters
- `value`: The new value.

`autocompletionOpenForNoText`()
If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(*value*)
Setter for *autocompletionOpenForNoText()*.

Parameters
- `value`: The new value.

setTextSelection(ifrom, ito)

Selects the specified part of the text.

Parameters
- `ifrom`: The selection begin index.
- `ito`: The selection end index.

textSelection()

Returns the current text selection indices.

OnChanged

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

OnPopupTextSelected

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters
- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namespace this widget contains to.

nameScope()
The own namespace (or the namespace it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(*value*)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(*value*)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(*value*)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(*value*)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(*value*)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(*value*)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(*value*)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(*value*)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(*value*)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(*value*)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(*value*)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(*value*)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(*value*)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(*value*)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class PasswordEditBox` : **`public`** *clove*::*EditBox*
A box for password input from the user.

Public Functions

`readOnly`()
If the edit box is read-only or editable by the user.

`setReadOnly`(value)
Setter for *readOnly()*.

Parameters
- `value`: The new value.

`text`()
The current text.

`setText`(value)
Setter for *text()*.

Parameters

- `value`: The new value.

`hintText`()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems`()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters
- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters
- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters
- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch (value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch ()
Alias for *horizontalStretchAffinity()*.

setHstretch (value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy ()
If the widget is in busy state, typically resulting in a loading animation.

setBusy (value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy ()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy (token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus ()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize ()
Returns inner width and height of this widget.

outerSize ()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class PopupMenu` : **`public`** *clove*::*AbstractMenu*

A popup menu (i.e. a box with vertically listed actions).

Public Functions

`expanderIndicatorDirection``()`

If the expander indicator is to be shown on the `'right'` or `'left'` side.

`setExpanderIndicatorDirection``(value)`

Setter for *expanderIndicatorDirection()*.

Parameters

- `value`: The new value.

`static show``(menu, showconfig)`

Shows a popup menu.

The root view of `menu` is expected to be a *clove::PopupMenu* (or a subclass).

Parameters

- `menu`: The widget configuration, as for *clove::build()*, which specifies the popup menu.
- `showconfig`: A configuration object which specifies some presentation aspects.

`actions``()`

The actions to be shown in this menu.

It is a list of action configuration objects, each having a structure like `{name:’myaction’, label:’My menu action’}. It can have a list of sub-items in the subactions` property.

Instead of a simple list, it may also be a *clove::Datasource* with the action configuration structures aligned in rows.

Use *clove::MenuSeparator* for a separator.

One action allows this properties:

- `name`: The action name.
- `label`: The label string.
- `icon`: A *clove::Icon*.
- `disabled`: If it is disabled.
- `invisible`: If it is invisible.

- `checkable`: If it is checkable.
- `checked`: If it is checked.

`setActions``(value)`
Setter for *actions()*.

Parameters
- `value`: The new value.

`OnActionTriggered`
Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

`OnBeforeSubactionsExpanded`
Triggered just before a branch of subaction is expanded.

Implement this event for populating it dynamically.

This is a *clove.Event* instance. See Manual for details.

`declareProperty``(k, defaultV)`
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty``(k)`
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

`setProperty``(k, v)`
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

`bindProperty``(k, vb)`
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init``(rootNameScope)`
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

class PopupMenuButton : **public** *clove*::*Button*
A special *clove::Button* which opens a popup menu when it is triggered.

Public Functions

actions()
The menu actions. Same as for *clove::Menubar::actions()*.

setActions(value)
Setter for *actions()*.

Parameters
- `value`: The new value.

OnActionTriggered
Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

label()
The label text.

setLabel(value)
Setter for *label()*.

Parameters
- `value`: The new value.

icon()
The optional *clove::Icon* button icon.

setIcon(value)
Setter for *icon()*.

Parameters
- `value`: The new value.

checkable()
If the button is checkable (i.e. has a checked-flag).

If it has, the checked-flag is controlled by *checked()*.

setCheckable(value)
Setter for *checkable()*.

Parameters
- `value`: The new value.

checked()
For a checkable button, return if it is checked.

See also *checkable()*.

User interactions do not toggle the checked flag by default. This must be scripted in own event handlers as needed.

setChecked(value)
Setter for *checked()*.

Parameters
- `value`: The new value.

OnClicked
Triggered when the user clicks on this button.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass` (clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass` (clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass` (clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned` (clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope` ()

The namescope this widget contains to.

`nameScope` ()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove` (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled` ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility` ()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets` ()

List of the children *clove::Widget* instances.

`parentWidget` ()

The parent *clove::Widget*.

`name` ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class ProgressBar` : **`public`** *clove*::*Widget*

A progress bar.

Public Functions

`value``()`

The progress value between `0.0` and `1.0`; or `undefined` for indeterminate progress.

`setValue``(`*value*`)`

Setter for *value()*.

Parameters

- `value`: The new value.

`orientation``()`

If to present the progress bar `'vertical'`ly or `'horizontal'`ly.

`setOrientation``(`*value*`)`

Setter for *orientation()*.

Parameters

- `value`: The new value.

`label``()`

An optional additional label text, which is displayed combined with the progress value.

`setLabel``(`*value*`)`

Setter for *label()*.

Parameters

- `value`: The new value.

`labelFunction``()`

A `function(value, widget)` which returns a custom label text.

`setLabelFunction``(`*value*`)`

Setter for *labelFunction()*.

Parameters

- `value`: The new value.

`declareProperty``(`k, defaultV`)`

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty``(k)`

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()
Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(*value*)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(*value*)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(*value*)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(*value*)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(*value*)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(*value*)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(*value*)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(*value*)

Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(*value*)

Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(*value*)

Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(*value*)

Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(*value*)

Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(*value*)

Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(*value*)

Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class ProxyDatasource` : **`public`** *clove*::*Datasource*, **`public`** *clove*::*Headersource*

A *clove::Datasource* implementation which proxies the content of another datasource in a somehow transformed way.

Note: This class is an abstract base class. Use one of the subclasses.

Subclassed by *clove::FilterProxyDatasource*, *clove::SortProxyDatasource*

Public Functions

`setDatasource`(datasource)

Sets the source datasource.

Parameters

- `datasource`: The source *clove::Datasource*.

`proxyPointerToNativePointer`(proxyptr)

Translates a *clove::DatasourceValuePointer* from this proxy datasource to a one for the source datasource.

Parameters

- `proxyptr`: A value pointer.

`nativePointerToProxyPointer`(nativeptr)

Translates a *clove::DatasourceValuePointer* from the source datasource to a one for this proxy datasource.

Parameters

- `nativeptr`: A value pointer.

`refresh`()

Refreshes the filtering.

Call this when the outer conditions changed and the filters must be applied again.

`getValue`(ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`getMetadata`(ptr)

Returns an object of metadata information (like icons, status infos used for styling, …). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`changeValue`(ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

`rowCount` (*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`columnCount` (*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`valuePointer` (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent` (ptr)

Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

`valuePointerNavigateInDepth` (ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

`OnDataInsert`

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`

Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)

Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader` (irow, *parent*)

Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible`(*parent*)
If row headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible`(*parent*)
If column headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`
Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`
Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`
Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`
Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

`class RadioButton` : **`public`** *clove*::*Widget*
A radio button.

Typically some radio buttons are assigned to one *clove::RadioGroup* so the user can choose one of them (while a *clove::CheckButton* allows 0..n choices).

Public Functions

`label`()
The label text.

`setLabel`(value)
Setter for *label()*.

Parameters
- `value`: The new value.

`checked`()
If this radio button is checked.

See also *clove::RadioGroup::selected()*.

`setChecked`(value)
Setter for *checked()*.

Parameters
- `value`: The new value.

`group`()
The *clove::RadioGroup* this radio button belongs to.

Note: It is possible to assign strings to it, see *clove::RadioGroup::getGroupByPublicName()*.

setGroup(value)
Setter for *group()*.

Parameters
- `value`: The new value.

groupValue()
The currently selected value of the radiogroup this button belongs to. See *group()*.

setGroupValue(value)
Setter for *groupValue()*.

Parameters
- `value`: The new value.

valueDef()
The value this button is representing. You can use this together with *clove::RadioGroup::selectedValue()*.

setValueDef(value)
Setter for *valueDef()*.

Parameters
- `value`: The new value.

OnChanged
Triggered when this radio button was selected.

This is a *clove.Event* instance. See Manual for details.

OnClicked
Triggered when this radio button was selected. Same as OnChanged.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

setProperty(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.

- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters
- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

class RadioGroup

Groups some *clove::RadioButton* together in a logical way, so the user can select exactly one of them.

Assign buttons to a group by setting *clove::RadioButton::group()*.

Public Functions

RadioGroup()

selected()
Returns the *clove::RadioButton* which is currently selected in this group.

select(w)
Selects a button in the group.

Parameters
- `w`: The *clove::RadioButton* to select.

OnChanged
Triggered when another button was selected in this group.

This is a *clove.Event* instance. See Manual for details.

static getGroupByPublicName(name)
Returns a *clove::RadioGroup* by name. Either returns an already existing one, or creates a new one.

Developers of library functionality should not use this function due to risks of naming clashes.
Parameters
- `name`: The group name.

selectedIndex()
Returns the position index of the currently selected child in this group.

selectedValue()
Returns the value of the currently selected child in this group.

selectByIndex(i)
Selects a child by its position index.

Parameters
- `i`: The position index.

selectByValue(v)
Selects a child by its value. It will also understand position indexes as fallback.

Parameters
- `v`: The value.

unselectAll()
Unselects all childs in this group.

class RawResizeSplitter : **public** *clove*::*Widget*
The actual splitter in a *clove::ResizeSplitter*.

Not intended for direct usage in a user interface!

Public Functions

OnMoveBegin
Triggered when the user begins to move the splitter.

This is a *clove.Event* instance. See Manual for details.

OnMove
Triggered subsequently while the user moves the splitter.

This is a *clove.Event* instance. See Manual for details.

`OnMoveEnd`
Triggered when the user ends moving the splitter.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

`setProperty`(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.
Parameters
- `rootNameScope`: The root namescope to add this widget to.

`doinit`()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class ResizeSplitter` : **`public`** *clove*::*Widget*

A container for multiple widgets with splitters for manual resizing between them.

Public Functions

`orientation`()

If the two widgets are stacked `'vertical'`ly or `'horizontal'`ly.

`setOrientation`(value)

Setter for *orientation()*.

Parameters

- `value`: The new value.

`body`()

The list of body widgets (i.e. the widgets to show separated by splitters) as widget configurations like for *clove::build()*.

`setBody`(value)

Setter for *body()*.

Parameters

- `value`: The new value.

`bodyWidgets`()

Returns the list of all body widgets as *clove::Widget* instances.

`setBodyWidget`(i, config)

Sets (overrides) the body widget at a given position.

Parameters

- `i`: The widget position.
- `config`: The widget as widget configurations like for *clove::build()*.

`addBodyWidget` (config)
Adds a new body widget to the end.

Parameters
- `config`: The widget as widget configurations like for *clove::build()*.

`insertBodyWidget` (i, config)
Inserts a new body widget somewhere.

Parameters
- `i`: The widget position.
- `config`: The widget as widget configurations like for *clove::build()*.

`sizeFractions` ()
The sizes of each widget inside this resize splitter as list of numbers. It has a positive entry for each widget, describing the amount of room it takes. The values have just relative meaning (and are typically fractions of 1).

`setSizeFractions` (value)
Setter for *sizeFractions()*.

Parameters
- `value`: The new value.

`showOnly` ()
Specifies if to show just one of the two sides (`0`, `1`, `undefined`).

`setShowOnly` (value)
Setter for *showOnly()*.

Parameters
- `value`: The new value.

`splitterWidth` ()
The splitter width (height for vertical orientation) as css length.

`setSplitterWidth` (value)
Setter for *splitterWidth()*.

Parameters
- `value`: The new value.

`splitterVisibility` ()
The visibility of the splitter. See *clove::Widget::visibility()* for details.

`setSplitterVisibility` (value)
Setter for *splitterVisibility()*.

Parameters
- `value`: The new value.

`declareProperty` (k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty` (k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters
- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

`addStyleClass` (clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

`removeStyleClass` (clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

`isStyleClass` (clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

`setStyleClassAssigned` (clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope` ()
The namescope this widget contains to.

`nameScope` ()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove` (removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.
Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled` ()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility` ()
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets` ()
List of the children *clove::Widget* instances.

`parentWidget` ()
The parent *clove::Widget*.

`name` ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class RichEdit` : **`public`** *clove*::*Widget*

A user input box for text with rich formatting.

This widget includes formatting toolbars. For a bare scriptable edit box, see *clove::RichEditBox*.

Public Functions

`htmlContent`()

The content text as html string.

`setHtmlContent`(value)

Setter for *htmlContent()*.

Parameters

- `value`: The new value.

`richeditbox`()

Returns the inner *clove::RichEditBox*.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy` (token)

Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

`hasFocus` ()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize` ()

Returns inner width and height of this widget.

`outerSize` ()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class RichEditBox` : **`public`** *clove*::*Widget*

A user input box for text with rich formatting.

This widget allows to be controlled from external toolbars. For a widget including a formatting toolbar, see *clove::RichEdit*.

Public Functions

`htmlContent`()
The content text as html string.

`setHtmlContent`(value)
Setter for *htmlContent()*.

Parameters
- `value`: The new value.

`selection`()
The current selection (as a web-api Range object).

`setSelection`(value)
Setter for *selection()*.

Parameters
- `value`: The new value.

`contentHtmlNode`()
Returns the html content dom node.

`toggleSelectionBold`()
Toggles the bold-flag for the current selection.

`toggleSelectionItalic`()
Toggles the italic-flag for the current selection.

`toggleSelectionUnderline`()
Toggles the underline-flag for the current selection.

`selectionIncreaseFontSize`()
Increases the font size for the current selection.

`selectionDecreaseFontSize`()
Decreases the font size for the current selection.

`selectionSetForegroundColor`(c)
Sets the foreground color for the current selection.

Parameters
- `c`: The color string.

`selectionSetBackgroundColor`(c)
Sets the background color for the current selection.

Parameters
- `c`: The color string.

`selectionInsertList`(config)
Inserts a list at the current place.

config may contain:
- `ordered`: If the list is ordered.
  Parameters
  - `config`: The list configuration.

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

class RootNameScope : **public** *clove*::*NameScope*
A standalone namescope.

Public Functions

getByName (name)
Finds a widget by name within this namescope.

Parameters
- `name`: The widget name.

addChildNameScope (childnamescope)
Adds a namescope to the children of this one, so a searcher will also search in this child.

Parameters
- `childnamescope`: The child namescope.

class ScrollView : **public** *clove*::*Widget*
A container for making the children scrollable.

Public Functions

body ()
The inner widget as widget configuration like for *clove::build()*.

setBody (value)
Setter for *body()*.

Parameters
- `value`: The new value.

bodyWidget ()
Returns the body as *clove::Widget*.

bodySize ()
The (outer) size of the body widget.

verticalScrollPosition ()
The vertical scroll position.

setVerticalScrollPosition (value)
Setter for *verticalScrollPosition()*.

Parameters

- `value`: The new value.

horizontalScrollPosition()

The horizontal scroll position.

setHorizontalScrollPosition(value)

Setter for *horizontalScrollPosition()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly``()`

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`getPreferredHeightForWidth``(w)`

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope``()`

The namescope this widget contains to.

`nameScope``()`

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Slider` : **`public`** *clove*::*Widget*

A slider allows the user to choose a number from a given value interval.

The user chooses a value by moving a handle alongside a line.

Public Functions

`min``()`

The minimum value.

`setMin``(`*value*`)`

Setter for *min()*.

Parameters

- `value`: The new value.

`max``()`

The maximum value.

`setMax``(`*value*`)`

Setter for *max()*.

Parameters

- `value`: The new value.

`stepsize``()`
The step size.

`setStepsize``(`*value*`)`
Setter for *stepsize()*.

Parameters
- `value`: The new value.

`value``()`
The current slider value.

`setValue``(`*value*`)`
Setter for *value()*.

Parameters
- `value`: The new value.

`orientation``()`
If to present the slider `'vertical'`ly or `'horizontal'`ly.

`setOrientation``(`*value*`)`
Setter for *orientation()*.

Parameters
- `value`: The new value.

`OnChanged`
Triggered when the slider value changed.

This is a *clove.Event* instance. See Manual for details.

`declareProperty``(`k, defaultV`)`
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty``(`k`)`
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

`setProperty``(`k, v`)`
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(*value*)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(*value*)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass``()`

Custom css class(es).

`setStyleClass``(`*value*`)`

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style``()`

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(`*value*`)`

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility``()`

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility``(`*value*`)`

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing``()`

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing``(`*value*`)`

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus``()`

If the widget can have the keyboard focus.

`setMayFocus``(`*value*`)`

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity``()`

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity``(`*value*`)`

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(*value*)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(*value*)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(*value*)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(*value*)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(*value*)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(*value*)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class SortProxyDatasource` : **`public`** *clove::ProxyDatasource*

A *clove::Datasource* implementation which sorts another *clove::Datasource*.

Public Functions

`SortProxyDatasource`(datasource, rowcomparefct, colcomparefct)

Parameters
- `datasource`: The source *clove::Datasource*.
- `rowcomparefct`: A `function(datasource, irow, jrow, parent)` returning a negative value if `irow` points to a row lower than `jrow`, a positive value if it is greater, or 0 if both are equal.
- `colcomparefct`: A `function(datasource, icol, jcol, parent)` returning a negative value if `icol` points to a column lower than `jcol`, a positive value if it is greater, or 0 if both are equal.

`setRowComparator`(rowcomparefct)

Sets the row comparator function.

Parameters
- `rowcomparefct`: The row comparator function. See constructor for details.

`setColumnComparator`(colcomparefct)

Sets the column comparator function.

Parameters
- `colcomparefct`: The column comparator function. See constructor for details.

`setDatasource`(datasource)

Sets the source datasource.

Parameters
- `datasource`: The source *clove::Datasource*.

`proxyPointerToNativePointer`(proxyptr)

Translates a *clove::DatasourceValuePointer* from this proxy datasource to a one for the source datasource.

Parameters
- `proxyptr`: A value pointer.

`nativePointerToProxyPointer`(nativeptr)

Translates a *clove::DatasourceValuePointer* from the source datasource to a one for this proxy datasource.

Parameters
- `nativeptr`: A value pointer.

`refresh`()

Refreshes the filtering.

Call this when the outer conditions changed and the filters must be applied again.

`getValue`(ptr)

Returns the value for a given node.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

`getMetadata`(ptr)

Returns an object of metadata information (like icons, status infos used for styling, . . .). Optional.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

changeValue (ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount (*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount (*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

parent (ptr)

Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

valuePointerNavigateInDepth (ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

OnDataInsert

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataRemove

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataUpdate

Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

getRowHeader (irow, *parent*)

Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader`(irow, *parent*)

Returns the header configuration for a given column.

Parameters
- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible`(*parent*)

If row headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible`(*parent*)

If column headers are visible.

Parameters
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

`class Spacer` : **`public`** *clove*::*Widget*

A spacer widget for filling gaps in a layout.

It is essentially just a widget which does nothing, but respects the layouting properties like all widgets. Add them to a layout and configure them according to your sizing requirements.

Public Functions

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing`()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class StackLayout` : **`public`** *clove*::*GridLayout*
A mixin for stack layout implementations.

Subclassed by *clove::HorizontalStack*, *clove::VerticalStack*

Public Functions

`insertRow`(i)
Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

`insertColumn`(i)
Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

`removeRow`(i)
Removes a row from the grid.

Parameters

- `i`: The row index.

`removeColumn`(i)
Removes a column from the grid.

Parameters

- `i`: The column index.

`children`()
List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)
Setter for *children()*.

Parameters

- `value`: The new value.

`addChild`(value)
Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds`()
Removes all childs from the layout.

class symbols
Namespace for symbol constants.

They make a few Unicode symbols accessible by a name.

class TableView : **public** *clove::DataView*
A view which shows a *clove::Datasource* in a table.

Public Functions

datasource()
The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

setDatasource(value)
Setter for *datasource()*.

Parameters
- `value`: The new value.

dataViewProcessor()
An optional `function(ptr, value)` for processing a datasource value to a display string.

setDataViewProcessor(value)
Setter for *dataViewProcessor()*.

Parameters
- `value`: The new value.

cellGenerator()
A generator function for complex cell content.

This method is called for each cell with `(clove::Widget, clove::DatasourceValuePointer)`. It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

setCellGenerator(value)
Setter for *cellGenerator()*.

Parameters
- `value`: The new value.

alwaysAllocateExpanderSpace()
If some space is allocated for an expander even for cells without children.

setAlwaysAllocateExpanderSpace(value)
Setter for *alwaysAllocateExpanderSpace()*.

Parameters
- `value`: The new value.

showOnlyFirstColumn()
If to show only the first column and hide the other ones.

setShowOnlyFirstColumn(value)
Setter for *showOnlyFirstColumn()*.

Parameters

- `value`: The new value.

`hideExpanders`()

If to hide all expanders.

`setHideExpanders`(value)

Setter for *hideExpanders()*.

Parameters

- `value`: The new value.

`allowSelection`()

If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

`setAllowSelection`(value)

Setter for *allowSelection()*.

Parameters

- `value`: The new value.

`allowChecking`()

If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

`setAllowChecking`(value)

Setter for *allowChecking()*.

Parameters

- `value`: The new value.

`granularity`()

The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

`setGranularity`(value)

Setter for *granularity()*.

Parameters

- `value`: The new value.

`showChangeMenu`()

If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:

- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

`setShowChangeMenu`(value)

Setter for *showChangeMenu()*.

Parameters

- `value`: The new value.

`editOnGesture`()

If the user shall be able to edit cells by double clicking/tapping them.

setEditOnGesture(value)
Setter for *editOnGesture()*.

Parameters
- `value`: The new value.

rowsResizable()
If the user shall be able to resize rows.

setRowsResizable(value)
Setter for *rowsResizable()*.

Parameters
- `value`: The new value.

columnsResizable()
If the user shall be able to resize columns.

setColumnsResizable(value)
Setter for *columnsResizable()*.

Parameters
- `value`: The new value.

selectCell(ptr)
Selects a cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

isCellSelected(ptr)
Checks if a given cell is selected.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

checkedCells()
Returns the checked cells as list of *clove::DatasourceValuePointer*.

setCheckedCells(ptrs)
Seturns the checked cells.

Parameters
- `ptrs`: A list of *clove::DatasourceValuePointer*.

isCellChecked(ptr)
Checks if a given cell is checked.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

setCellChecked(ptr, val)
Sets if a given cell is checked.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

selection()
Returns the selected item as *clove::DatasourceValuePointer*.

headersource()
The *clove::Headersource* providing row and column header configurations.

setHeadersource(value)
Setter for *headersource()*.

Parameters
- `value`: The new value.

gridVisible()
If to show a cell grid.

setGridVisible(value)
Setter for *gridVisible()*.

Parameters
- `value`: The new value.

nodeActivationNeedsDoubleClick()
If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

setNodeActivationNeedsDoubleClick(value)
Setter for *nodeActivationNeedsDoubleClick()*.

Parameters
- `value`: The new value.

expandCell(ptr)
Expands a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

expandCellRecursive(ptr)
Expands a given cell and all parents.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

collapseCell(ptr)
Collapses a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

isCellExpanded(ptr)
Checks if a given cell is expanded.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

editCell(ptr)
Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.
Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

OnSelectionChanged
Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class TabView` : **`public`** *clove*::*Widget*

A tabbed container.

Typically used for grouping and folding some user interface parts together for better space efficiency and higher usability.

Subclassed by *clove::Carousel*

Public Functions

`tabs``()`

The list of tabs.

Each tab is a widget configuration, like for *clove::build()*, with some additional optional keys which holds infos like the tab header text. So, for example, one item in that list could be `{view:'Something', …, tabLabel:'Tab1'}`. See also *addTab()*.

`setTabs``(value)`

Setter for *tabs()*.

Parameters

- `value`: The new value.

`currentTab``()`

The index of the currently selected tab.

`setCurrentTab``(value)`

Setter for *currentTab()*.

Parameters

- `value`: The new value.

`userMayAddTabs``()`

If to show a button for adding new tabs.

If `true`, implement a handler for OnTabCreationRequested and create a tab with *addTab()* there.

`setUserMayAddTabs``(value)`

Setter for *userMayAddTabs()*.

Parameters
- `value`: The new value.

`tabBarLocation`()
Where to place the tab bar.

Either `'top'`, `'left'`, `'bottom'` or `'right'`.

`setTabBarLocation`(value)
Setter for *tabBarLocation()*.

Parameters
- `value`: The new value.

`switchInvisibleAnimationName`()
Optional animation name for switching away from a tab.

`setSwitchInvisibleAnimationName`(value)
Setter for *switchInvisibleAnimationName()*.

Parameters
- `value`: The new value.

`switchVisibleAnimationName`()
Optional animation name for switching to a tab.

`setSwitchVisibleAnimationName`(value)
Setter for *switchVisibleAnimationName()*.

Parameters
- `value`: The new value.

`switchInvisibleAnimationDuration`()
Optional animation duration (in msec) for the switch away animation.

See also *clove::TabView::switchInvisibleAnimationName()*.

`setSwitchInvisibleAnimationDuration`(value)
Setter for *switchInvisibleAnimationDuration()*.

Parameters
- `value`: The new value.

`switchVisibleAnimationDuration`()
Optional animation duration (in msec) for the switch animation.

See also *clove::TabView::switchVisibleAnimationName()*.

`setSwitchVisibleAnimationDuration`(value)
Setter for *switchVisibleAnimationDuration()*.

Parameters
- `value`: The new value.

`OnTabCreationRequested`
Triggered when the user requested a new tab.

See also *userMayAddTabs()*.

This is a *clove.Event* instance. See Manual for details.

`addTab`(config)
Adds a new tab.

The configuration may contain the following additional keys:

- `tabLabel`: The tab header text.
- `tabIcon`: The tab icon as *clove::Icon*.
- `tabMayBeClosedByUser`: If the user may close this tab.

This method returns a *clove::Widget* which can be used for getting subwidgets or removing the tab.

Parameters

- `config`: A widget configuration like for *clove::build()*.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters
- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class TimeBox` : **`public`** *clove*::*EditBox*

A user input box for time input.

The actual behavior depends on the browser.

Public Functions

`time`()

The time value in the box as JavaScript Date.

`setTime`(value)

Setter for *time()*.

Parameters

- `value`: The new value.

`readOnly`()

If the edit box is read-only or editable by the user.

`setReadOnly`(value)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

`text`()

The current text.

`setText`(value)

Setter for *text()*.

Parameters

- `value`: The new value.

`hintText`()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems`()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()
List of the children *clove::Widget* instances.

`parentWidget`()
The parent *clove::Widget*.

`name`()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)
Setter for *name()*.

Parameters
- `value`: The new value.

`enabled`()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

`styleClass`()
Custom css class(es).

`setStyleClass`(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

`style`()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)
Setter for *style()*.

Parameters
- `value`: The new value.

`visibility`()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

class Toolbar : **public** *clove*::*Widget*
A toolbar has header texts and a menu bar in a perceptible visual bar, typically used on top of a user interface with all available width.

Public Functions

actions()
Menu actions.

See *clove::Menubar::actions()*.

setActions(value)
Setter for *actions()*.

Parameters
- `value`: The new value.

head1()
The 1st header text.

setHead1(value)
Setter for *head1()*.

Parameters
- `value`: The new value.

head2()
The 2nd header text.

setHead2(value)
Setter for *head2()*.

Parameters
- `value`: The new value.

headControl()
An optional widget for arbitrary control purposes as widget configuration like for *clove::build()*. It's displayed below the menu bar in full width.

setHeadControl (value)
Setter for *headControl()*.

Parameters
- `value`: The new value.

headControlWidget ()
Returns the control widget as *clove::Widget* (or undefined if no *headControl()* is set).

icon ()
The header icon as *clove::Icon*.

setIcon (value)
Setter for *icon()*.

Parameters
- `value`: The new value.

OnActionTriggered
Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

setProperty (k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()

Sets the focus to this widget.

`isAlive`()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth`()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth`()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters
- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters
- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters
- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters
- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing``()`
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

`setStrictHorizontalSizing``(value)`
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing``()`
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing``(value)`
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

`vstretch``()`
Alias for *verticalStretchAffinity()*.

`setVstretch``(value)`
Setter for *vstretch()*.

Parameters
- `value`: The new value.

`hstretch``()`
Alias for *horizontalStretchAffinity()*.

`setHstretch``(value)`
Setter for *hstretch()*.

Parameters
- `value`: The new value.

`busy``()`
If the widget is in busy state, typically resulting in a loading animation.

`setBusy``(value)`
Setter for *busy()*.

Parameters
- `value`: The new value.

`registerBusy``()`
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy``(token)`
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

class TreeView : **public** *clove*::*DataView*

A view which shows a *clove::Datasource* in a tree.

Public Functions

`datasource`()

The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

`setDatasource`(value)

Setter for *datasource()*.

Parameters

- `value`: The new value.

`dataViewProcessor`()

An optional `function(ptr, value)` for processing a datasource value to a display string.

`setDataViewProcessor`(value)

Setter for *dataViewProcessor()*.

Parameters

- `value`: The new value.

`cellGenerator`()

A generator function for complex cell content.

This method is called for each cell with `(clove::Widget, clove::DatasourceValuePointer)`. It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

`setCellGenerator`(value)

Setter for *cellGenerator()*.

Parameters

- `value`: The new value.

`alwaysAllocateExpanderSpace`()

If some space is allocated for an expander even for cells without children.

`setAlwaysAllocateExpanderSpace`(value)

Setter for *alwaysAllocateExpanderSpace()*.

Parameters

- `value`: The new value.

`showOnlyFirstColumn`()

If to show only the first column and hide the other ones.

`setShowOnlyFirstColumn`(value)

Setter for *showOnlyFirstColumn()*.

Parameters

- `value`: The new value.

`hideExpanders`()

If to hide all expanders.

`setHideExpanders`(value)

Setter for *hideExpanders()*.

Parameters

- `value`: The new value.

`allowSelection`()

If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

`setAllowSelection`(value)

Setter for *allowSelection()*.

Parameters

- `value`: The new value.

`allowChecking`()

If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

`setAllowChecking`(value)

Setter for *allowChecking()*.

Parameters

- `value`: The new value.

`granularity`()

The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

`setGranularity`(value)

Setter for *granularity()*.

Parameters

- `value`: The new value.

`showChangeMenu`()

If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:

- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

`setShowChangeMenu`(value)

Setter for *showChangeMenu()*.

Parameters

- `value`: The new value.

`editOnGesture`()

If the user shall be able to edit cells by double clicking/tapping them.

`setEditOnGesture`(value)

Setter for *editOnGesture()*.

Parameters

- `value`: The new value.

`rowsResizable`()

If the user shall be able to resize rows.

setRowsResizable(value)
Setter for *rowsResizable()*.

Parameters
- `value`: The new value.

columnsResizable()
If the user shall be able to resize columns.

setColumnsResizable(value)
Setter for *columnsResizable()*.

Parameters
- `value`: The new value.

selectCell(ptr)
Selects a cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

isCellSelected(ptr)
Checks if a given cell is selected.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

checkedCells()
Returns the checked cells as list of *clove::DatasourceValuePointer*.

setCheckedCells(ptrs)
Seturns the checked cells.

Parameters
- `ptrs`: A list of *clove::DatasourceValuePointer*.

isCellChecked(ptr)
Checks if a given cell is checked.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

setCellChecked(ptr, val)
Sets if a given cell is checked.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

selection()
Returns the selected item as *clove::DatasourceValuePointer*.

headersource()
The *clove::Headersource* providing row and column header configurations.

setHeadersource(value)
Setter for *headersource()*.

Parameters
- `value`: The new value.

gridVisible()
If to show a cell grid.

setGridVisible (value)
Setter for *gridVisible()*.

Parameters
- `value`: The new value.

nodeActivationNeedsDoubleClick ()
If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

setNodeActivationNeedsDoubleClick (value)
Setter for *nodeActivationNeedsDoubleClick()*.

Parameters
- `value`: The new value.

expandCell (ptr)
Expands a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

expandCellRecursive (ptr)
Expands a given cell and all parents.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

collapseCell (ptr)
Collapses a given cell.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

isCellExpanded (ptr)
Checks if a given cell is expanded.

Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

editCell (ptr)
Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.
Parameters
- `ptr`: The *clove::DatasourceValuePointer*.

OnSelectionChanged
Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()
Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.
Parameters
- `w`: The width in pixel.

`getPreferredHeightForWidth`(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

name ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled ()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass ()
Custom css class(es).

setStyleClass (value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style ()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle (value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility ()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility (value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing ()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing (value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class utils`
Namespace for some utilities.

Public Functions

`main_parts_resized`(reason)
Call this method if resizing is required due to external situations.

It is not required to call this method in typical situations. A situation which requires this method is changing the stylesheets from outside.

Parameters

- `reason`: Optional arbitrary object which describes the reason for resizing. Can be evaluated by handlers in a custom way.

`applyDefaultsToConfig`(config, defaults)
Returns a copy of `config` with the additional members from `defaults`, whereever there was no value in `config`.

Parameters

- `config`: The source object.
- `defaults`: Default values for members to apply to the result whereever they are missing.

`addOnDestroyHandler`(domnode, handler)
Listen for the removal of a node in the dom tree and eventually execute a handler function.

Parameters

- `domnode`: The node in the dom tree to observe.
- `handler`: The function to execute on removal of `domnode`.

`addOnDragHandler`(domnode, dragbeginhandler, draghandler, dragendhandler)
Listen for a drag event of a node and eventually execute handler functions.

A drag consists of mouse/finger down, moving and releasing.

Parameters

- `domnode`: The node in the dom tree to observe.
- `dragbeginhandler`: The function to execute when a drag operation begins.
- `draghandler`: The function to execute while moving (with relative coordinates as parameters).
- `dragendhandler`: The function to execute when a drag operation ends.

`addOnDoubleClickTapHandler`(domnode, handler)
Listen for double click/tap event of a node and eventually execute a handler function.

This is like a pure double click event, but also supports touch.

See also *clove::utils::removeOnDoubleClickTapHandler()*.

Parameters

- `domnode`: The node in the dom tree to observe.
- `handler`: The function to execute on double click/tap.

`removeOnDoubleClickTapHandler`(domnode, handler)

Remove a double click/tap event handler.

Parameters
- `domnode`: The node in the dom tree to remove a handler from.
- `handler`: The function to remove.

`arraysum`(x)

Computes the sum of all elements in an Array.

Parameters
- `x`: The Array to sum up.

`findWidgetForDomNode`(node)

Finds the (most inner) *clove::Widget* a given dom node contains to.

Parameters
- `node`: The dom node.

`getExtraSize`(node, withMargin)

Computes the extra size of a dom node.

This is the difference between inner size and the outer size (i.e. borders, padding, . . .).

Parameters
- `node`: The dom node to measure.
- `withMargin`: If to include margins.

`getInnerSize`(node)

Computes the inner size of a dom node.

This is the size without borders, padding, . . .

Parameters
- `node`: The dom node to measure.

`getOuterSize`(node, withMargin)

Computes the outer size of a dom node.

This is the size including borders, padding, . . .

Parameters
- `node`: The dom node to measure.
- `withMargin`: If to include margins.

`suspendResizing`()

Temporarily suspends all user interface resizing.

This is useful for performance reasons during intensive changes to the user interface.

Use this with caution and always call *clove::utils::resumeResizing* afterwards.

Return A token object used for resuming later on.

`resumeResizing`(token)

Resumes the user interface resizing after it was suspended with *clove::utils::suspendResizing*.

It automatically takes care of resizing everything which diverged meanwhile.

Parameters
- `token`: The token object.

`cssLengthToPixels`(v, axis)

Translates a css length specification string to a numeric pixel value.

Note: This does not make sense for all kinds of lengths, e.g. percentages will not work.

Parameters

- `v`: The css length.
- `axis`: The axis ('x' or 'y').

`insertToArray` (array, i, v)

Inserts a value into an array.

Parameters

- `array`: The array.
- `i`: The position.
- `v`: The new value to insert.

`popup` (config, showconfig)

Shows a popup.

This can show an arbitrary widget decoupled from the main user interface (swimming above it).

This call retuns a dialog result structure, containing the following:

- `widget`: The popup widget as *clove::Widget*.
- `namescope`: The root namescope for the popup as *clove::NameScope*.

  Parameters

  - `config`: The widget configuration, as for *clove::build()*, which specifies the popup.
  - `showconfig`: A configuration object which specifies some presentation aspects.

`isDescendantOf` (d, p)

Checks if `d` is a descendant of `p` in the html dom node.

Parameters

- `d`: An html dom node.
- `p`: An html dom node.

`arrayToDatasource` (array)

Builds a *clove::Datasource* containing the same content as the input array.

Parameters

- `array`: The input array.

`connectDatasourceToWidget` (widget, datasource, dspropname, rootptrpropname, inserthandlername, removehandlername, updatehandlername, afterdisconnectedfct, beforeconnectedfct, afterconnectedfct)

Connects a *clove::Datasource* to an internal widget interface (also removes the old connection).

Connects a datasource to a widget.

This method is solely used by the inner implementation of a widget for using a datasource. It is not useful to call it in any other situation! Used internally in widget implementations.

Parameters

- `widget`: A *clove::Widget*.
- `datasource`: A *clove::Datasource*.
- `dspropname`: The property name for storing the current datasource in the widget.
- `rootptrpropname`: The property name to get the root *clove::DatasourceValuePointer* from the widget.
- `inserthandlername`: The name of the widget's insert handler method.
- `removehandlername`: The name of the widget's remove handler method.
- `updatehandlername`: The name of the widget's update handler method.
- `afterdisconnectedfct`: Optional function called just after disconnection of the old datasource.
- `beforeconnectedfct`: Optional function called just before connecting the new datasource.

- `afterconnectedfct`: Optional function called just after connecting the new datasource.

Parameters

- `widget`: The *clove::Widget*.
- `datasource`: The *clove::Datasource* to connect.
- `dspropname`: The name of the datasource property.
- `rootptrpropname`: The name of the root value pointer property.
- `inserthandlername`: The name of the insert handler.
- `removehandlername`: The name of the remove handler.
- `updatehandlername`: The name of the update handler.
- `afterdisconnectedfct`: This function is called after the last datasource is disconnected.
- `beforeconnectedfct`: This function is called just before the datasource is connected.
- `afterconnectedfct`: This function is called right after the datasource is connected.

animateNode(node, animationname, animationduration, animationendedfct)

Plays a css animation on a html dom node for one iteration and fires a callback afterwards.

Parameters

- `node`: The html dom node to animate.
- `animationname`: The css animation name.
- `animationduration`: The animation duration in milliseconds.
- `animationendedfct`: a callback `function()` called when the animation ends.

setStyleClassAssigned(node, clss, assigned)

Sets or unsets a css class to an html dom node.

Parameters

- `node`: The html dom node to modify.
- `clss`: The css class name.
- `assigned`: If to assign or unassign it.

deferLoading(eload)

Defers loading of Clove by an event.

Used only internally and in exotic situations.

Call this before loading is finished (i.e. typically while the document itself loads) and before the `eload` event triggers (i.e. not when it already might have triggered).

See also *runOnLoaded()*.

Parameters

- `eload`: The *clove::Event* object which is triggered when it's ready for continue loading.

runOnLoaded(fct)

Runs a function as soon as possible, but not before Clove has finished loading.

Used only internally and in exotic situations. One would typically use *clove::populateUI()* instead.

Parameters

- `fct`: The `function()` to execute when Clove is loaded.

getFullPathForLoadedResource(tgtname, elementtype, elementsrcattrname)

Returns the full path for an already loaded script or stylesheet with known file name.

Used only internally and in exotic situations.

Parameters

- `tgtname`: The complete file name (bare; without slashes) of the file to search for.
- `elementtype`: Optional element type (default: `"script"`).
- `elementsrcattrname`: Optional element type's attribute name for the source url (if type is not `"script"` or `"style"`).

tryLoadScript (scriptpath, config)
Tries to load another javascript.

If it succeeds, it returns the url of it.

config may contain:
- `rootrefscript`: For relative script paths, this reference script is used in order to determine the base directory.
- `onload`: A `function()` to execute when the script is loaded.
  Parameters
  - `scriptpath`: The path to the javascript file.
  - `config`: A (optional) configuration object.

tryLoadStyle (stylepath, config)
Tries to load a css stylesheet.

If it succeeds, it returns the url of it.

config may contain:
- `rootrefscript`: For relative style paths, this reference script is used in order to determine the base directory.
- `onload`: A `function()` to execute when the stylesheet is loaded.
  Parameters
  - `stylepath`: The path to the css stylesheet file.
  - `config`: A (optional) configuration object.

OnMainResized
Triggered when the main view resized.

This is a *clove.Event* instance. See Manual for details.

class VerticalStack : **public** *clove*::*Widget*, **public** *clove*::*StackLayout*
A container which stacks child widgets row-wise.

Public Functions

rows ()
The list of child widgets as widget configurations like for *clove::build()*.

setRows (value)
Setter for *rows()*.

Parameters
- `value`: The new value.

declareProperty (k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

setProperty(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.
Parameters
- `rootNameScope`: The root namescope to add this widget to.

doinit()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()
Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass` (clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass` (clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass` (clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned` (clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope` ()

The namescope this widget contains to.

`nameScope` ()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove` (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled` ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility` ()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets` ()

List of the children *clove::Widget* instances.

`parentWidget` ()

The parent *clove::Widget*.

`name` ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters
- `value`: The new value.

enabled ()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)
Setter for *enabled()*.

Parameters
- `value`: The new value.

styleClass ()
Custom css class(es).

setStyleClass (value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style ()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle (value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility ()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility (value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing ()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing (value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus ()
If the widget can have the keyboard focus.

setMayFocus (value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`insertRow`(i)

Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

`insertColumn`(i)

Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

`removeRow`(i)

Removes a row from the grid.

Parameters

- `i`: The row index.

`removeColumn`(i)

Removes a column from the grid.

Parameters

- `i`: The column index.

`children`()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)

Setter for *children()*.

Parameters

- `value`: The new value.

`addChild`(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds`()

Removes all childs from the layout.

`class VideoPlayer` : **`public`** *clove*::*MediaPlayer*

A widget which plays a video source and optionally allows user controls.

Public Functions

static canPlayType(t)
Determines if the browser can play a certain video type and returns a string as described in the html specs (->"canPlayType").

Parameters
- `t`: A mimetype string.

autoPlay()
If to automatically start playback as soon as possible.

setAutoPlay(value)
Setter for *autoPlay()*.

Parameters
- `value`: The new value.

showControls()
If to show user controls for play, pause and more.

setShowControls(value)
Setter for *showControls()*.

Parameters
- `value`: The new value.

currentMediaPosition()
The current media playback position in seconds.

setCurrentMediaPosition(value)
Setter for *currentMediaPosition()*.

Parameters
- `value`: The new value.

loop()
If to playback in an endless loop.

setLoop(value)
Setter for *loop()*.

Parameters
- `value`: The new value.

muted()
If to mute the audio playback.

setMuted(value)
Setter for *muted()*.

Parameters
- `value`: The new value.

preload()
If to begin loading the media data as soon as possible.

setPreload(value)
Setter for *preload()*.

Parameters
- `value`: The new value.

volume()
The audio playback volume between `0.0` and `1.0`.

setVolume(value)
Setter for *volume()*.

Parameters
- `value`: The new value.

source()
The media source URL.

setSource(value)
Setter for *source()*.

Parameters
- `value`: The new value.

duration()
The duration of the loaded media source in seconds.

hasEnded()
If the playback has ended (i.e. reached the end).

isPaused()
If the playback is logically paused.

This does not return `true` just when loading stalls.

nativeNode()
Returns the native html dom node.

play()
Starts playback.

pause()
Pauses playback.

OnLoadingAborted
Triggered when the media loading aborted for some reasons (e.g. network errors).

This is a *clove.Event* instance. See Manual for details.

OnReadyForPlayback
Triggered when there are enough data for starting playback.

This is a *clove.Event* instance. See Manual for details.

OnDurationChanged
Triggered when the playback duration changed.

See also *duration()*.

This is a *clove.Event* instance. See Manual for details.

OnHasEnded
Triggered when the playback has ended.

See also *hasEnded()*.

This is a *clove.Event* instance. See Manual for details.

OnIsPaused
Triggered when the playback logically paused.

This is not triggered when loading stalls.

This is a *clove.Event* instance. See Manual for details.

`OnIsPlaying`

Triggered when the playback logically starts.

This is not triggered when loading has stalled and resumes.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters
- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled``()`
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility``()`
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets``()`
List of the children *clove::Widget* instances.

`parentWidget``()`
The parent *clove::Widget*.

`name``()`
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName``(value)`
Setter for *name()*.

Parameters
- `value`: The new value.

`enabled``()`
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled``(value)`
Setter for *enabled()*.

Parameters
- `value`: The new value.

`styleClass``()`
Custom css class(es).

`setStyleClass``(value)`
Setter for *styleClass()*.

Parameters
- `value`: The new value.

`style``()`
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(value)`
Setter for *style()*.

Parameters
- `value`: The new value.

`visibility``()`
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters
- `value`: The new value.

`strictHorizontalSizing`()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters
- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters
- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Widget`

Base class for a Clove widget.

Read the Manual about widgets.

Subclassed by *clove::AbstractMenu*, *clove::Border*, *clove::Button*, *clove::CheckButton*, *clove::DataView*, *clove::Dialog*, *clove::EditBox*, *clove::Expander*, *clove::FlatView*, *clove::Form*, *clove::Grid*, *clove::HorizontalStack*, *clove::HtmlView*, *clove::IconView*, *clove::ImageView*, *clove::Label*, *clove::MainView*, *clove::MediaPlayer*, *clove::ModalPanel*, *clove::ProgressBar*, *clove::RadioButton*, *clove::RawResizeSplitter*, *clove::ResizeSplitter*, *clove::RichEdit*, *clove::RichEditBox*, *clove::ScrollView*, *clove::Slider*, *clove::Spacer*, *clove::TabView*, *clove::Toolbar*, *clove::VerticalStack*, *clove::Wrap*

Public Functions

`Widget`(config, domnode)

Note: All widgets have a constructor with this signature. The configuration parameters are stored in `config`.

It is typically not required to call those constructors directly. Use *clove::build()* or some other factory functions (for special situations) instead.

Parameters

- `config`: A widget configuration as in *clove::build()*.
- `domnode`: The dom node to use as host for this widget.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.
Parameters
- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters
- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters
- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters
- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters
- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`class Wrap` : **`public`** *clove*::*Widget*, **`public`** *clove*::*WrapLayout*
A container for aligning child widgets in horizontal wrapping lines.

Public Functions

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.
Parameters
- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.
Parameters
- `k`: The widget property name.

`setProperty`(k, v)
General-purpose setter for widget properties.

See *getProperty()*.
Parameters
- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters
- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters
- `rootNameScope`: The root namescope to add this widget to.

doinit ()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()
Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()
Sets the focus to this widget.

isAlive ()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth ()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters
- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters
- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters
- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters
- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters
- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters
- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`children`()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)

Setter for *children()*.

Parameters

- `value`: The new value.

`addChild`(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()

Removes all childs from the layout.

class WrapLayout : **public** *clove*::*Layout*

A mixin for wrap layout implementations.

Subclassed by *clove::Wrap*

Public Functions

children()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)

Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()

Removes all childs from the layout.

18.1.2 Class clove::AbstractMenu

class *clove*::**AbstractMenu** : **public** *clove*::*Widget*

A base class for widgets which present a menu of actions to the user.

Subclassed by *clove::Menubar*, *clove::PopupMenu*

Public Functions

actions()

The actions to be shown in this menu.

It is a list of action configuration objects, each having a structure like `{name:'myaction', label:'My menu action'}. `It can have a list of sub-items in the` subactions` property.

Instead of a simple list, it may also be a *clove::Datasource* with the action configuration structures aligned in rows.

Use *clove::MenuSeparator* for a separator.

One action allows this properties:

- `name`: The action name.
- `label`: The label string.
- `icon`: A *clove::Icon*.
- `disabled`: If it is disabled.
- `invisible`: If it is invisible.

- `checkable`: If it is checkable.
- `checked`: If it is checked.

setActions (value)

Setter for *actions()*.

Parameters

- `value`: The new value.

OnActionTriggered

Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

OnBeforeSubactionsExpanded

Triggered just before a branch of subaction is expanded.

Implement this event for populating it dynamically.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be ‘undefined’ for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()

Sets the focus to this widget.

isAlive ()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth ()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()
The namescope this widget contains to.

`nameScope`()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()
List of the children *clove::Widget* instances.

`parentWidget`()
The parent *clove::Widget*.

`name`()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)
Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus``()`
If the widget can have the keyboard focus.

`setMayFocus``(value)`
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity``()`
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity``(value)`
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity``()`
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity``(value)`
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing``()`
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing``(value)`
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing``()`
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing``(value)`
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch``()`
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.3 Class clove::AjaxAsyncDatasource

`class` *`clove`*`::`**`AjaxAsyncDatasource`** : **`public`** *clove*::*AsyncDatasource*, **`public`** *clove*::*Headersource*

A *clove::AsyncDatasource* implementation which uses Ajax requests for data retrieval and modification.

Public Functions

`AjaxAsyncDatasource`(config)

The ajax configuration may contain:

- `data_pullUrl`: The url for data retrieval.
- `data_pushUrl`: The url for data modification.
- `data_pullParams`: Additional parameters for data retrieval.
- `data_pushParams`: Additional parameters for data modification.
- `data_params`: Additional parameters.
- `data_mayHaveChildren`: If the data items might have children.
- `data_autoPullCharily`: If automatic data fetching should be less aggressive.
- `params`: Additional parameters.
- `rootType`: The root type. This controls the prefix of the url and parameter keys to consider
- `ajaxAdditionalArgs`: Additional arguments for the ajax request.
- `answerIdKeyName`: The key name for the identification column used in reading server answers.
- `requestIdKeyName`: The key name for the identification used in requests.

- `headerConfigs`: Additional static header configurations. for fetching the first (and often only) level of objects. Instead of `data_`, other prefixes are allowed as well. The return type name of an retrieved object controls which prefix to use for fetching child objects.

  Parameters

  - `config`: The Ajax configuration.

`OnDataArrived`

Triggered when new data arrived from the server.

This is a *clove.Event* instance. See Manual for details.

`getAjaxId`(ptr)

Returns the Ajax id for a data cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`do_async_pull`(ptr, requestConfig)

Executes an asynchronous data modification.

Do not call this from outside, use *clove::AsyncDatasource::async_pull()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

`do_async_push`(ptr, v, requestConfig)

Executes an asynchronous data retrieval.

Do not call this from outside, use *clove::AsyncDatasource::async_push()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

`async_pull`(ptr, requestConfig)

Asynchronously modifies a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_pull()* instead.

`requestConfig` may contain:

- `onsuccess`: A `function()` called when the operation is completed and reflected in this datasource.
- `onfailed`: A `function(error)` called when an error occurred.
- `reason`: An optional custom object which can be used by observers for custom decisions.

  Parameters

  - `ptr`: The *clove::DatasourceValuePointer*.

  - `requestConfig`: The request configuration.

async_push (ptr, v, requestConfig)

Asynchronously retrieves a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_push()* instead.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

clearCache ()

Clears the internal caches.

getValue (ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata (ptr)

Returns an object of metadata information (like icons, status infos used for styling, . . .). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

changeValue (ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount (*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount (*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent` (ptr)
Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

`valuePointerNavigateInDepth` (ptr, direction, mayexpandfct)
Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

`OnDataInsert`
Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`
Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`
Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)
Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader` (irow, *parent*)
Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible` (*parent*)
If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible`(*parent*)

If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

18.1.4 Class clove::AsyncDatasource

`class` *`clove`*`::`**`AsyncDatasource`** : **`public`** *clove*::*Datasource*, **`public`** *clove*::*Headersource*

A *clove::Datasource* implementation which supports asynchronous data operations like Ajax requests.

Subclassed by *clove::AjaxAsyncDatasource*

Public Functions

`AsyncDatasource`(config)

The configuration may contain:

- `'errorhandler'`: Optional `function(error)` as fallback error handler.
- `'cacheAge'`: The maximum cache age in seconds.

Parameters

- `config`: The async datasource configuration.

`do_async_pull`(ptr, requestConfig)

Executes an asynchronous data modification.

Do not call this from outside, use *clove::AsyncDatasource::async_pull()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

`do_async_push`(ptr, v, requestConfig)

Executes an asynchronous data retrieval.

Do not call this from outside, use *clove::AsyncDatasource::async_push()* instead.

Override this method in custom implementations.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

`async_pull`(ptr, requestConfig)

Asynchronously modifies a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_pull()* instead.

`requestConfig` may contain:

- `onsuccess`: A `function()` called when the operation is completed and reflected in this datasource.
- `onfailed`: A `function(error)` called when an error occurred.
- `reason`: An optional custom object which can be used by observers for custom decisions.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `requestConfig`: The request configuration.

`async_push`(ptr, v, requestConfig)

Asynchronously retrieves a data value.

Do not override this in custom implementations, use *clove::AsyncDatasource::do_async_push()* instead.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `v`: The new value.
- `requestConfig`: The request configuration. See *clove::AsyncDatasource::async_pull()* for details.

clearCache()

Clears the internal caches.

getValue(ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata(ptr)

Returns an object of metadata information (like icons, status infos used for styling, …). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

changeValue(ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount(*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount(*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer(irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

parent (ptr)
Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

valuePointerNavigateInDepth (ptr, direction, mayexpandfct)
Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

OnDataInsert
Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataRemove
Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataUpdate
Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

getRowHeader (irow, *parent*)
Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

getColumnHeader (irow, *parent*)
Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

rowHeadersVisible (*parent*)
If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

columnHeadersVisible (*parent*)
If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

18.1.5 Class clove::AudioPlayer

`class` `clove::`**`AudioPlayer`** : **`public`** *clove::MediaPlayer*

A widget which plays an audio source and optionally allows user controls.

Public Functions

`static canPlayType``(t)`

Determines if the browser can play a certain audio type and returns a string as described in the html specs (->"canPlayType").

Parameters

- `t`: A mimetype string.

`autoPlay``()`

If to automatically start playback as soon as possible.

`setAutoPlay``(value)`

Setter for *autoPlay()*.

Parameters

- `value`: The new value.

`showControls``()`

If to show user controls for play, pause and more.

`setShowControls``(value)`

Setter for *showControls()*.

Parameters

- `value`: The new value.

currentMediaPosition()

The current media playback position in seconds.

setCurrentMediaPosition(value)

Setter for *currentMediaPosition()*.

Parameters

- `value`: The new value.

loop()

If to playback in an endless loop.

setLoop(value)

Setter for *loop()*.

Parameters

- `value`: The new value.

muted()

If to mute the audio playback.

setMuted(value)

Setter for *muted()*.

Parameters

- `value`: The new value.

preload()

If to begin loading the media data as soon as possible.

setPreload(value)

Setter for *preload()*.

Parameters

- `value`: The new value.

volume()

The audio playback volume between `0.0` and `1.0`.

setVolume(value)

Setter for *volume()*.

Parameters

- `value`: The new value.

source()

The media source URL.

setSource(value)

Setter for *source()*.

Parameters

- `value`: The new value.

duration()
The duration of the loaded media source in seconds.

hasEnded()
If the playback has ended (i.e. reached the end).

isPaused()
If the playback is logically paused.

This does not return `true` just when loading stalls.

nativeNode()
Returns the native html dom node.

play()
Starts playback.

pause()
Pauses playback.

OnLoadingAborted
Triggered when the media loading aborted for some reasons (e.g. network errors).

This is a *clove.Event* instance. See Manual for details.

OnReadyForPlayback
Triggered when there are enough data for starting playback.

This is a *clove.Event* instance. See Manual for details.

OnDurationChanged
Triggered when the playback duration changed.

See also *duration()*.

This is a *clove.Event* instance. See Manual for details.

OnHasEnded
Triggered when the playback has ended.

See also *hasEnded()*.

This is a *clove.Event* instance. See Manual for details.

OnIsPaused
Triggered when the playback logically paused.

This is not triggered when loading stalls.

This is a *clove.Event* instance. See Manual for details.

OnIsPlaying
Triggered when the playback logically starts.

This is not triggered when loading has stalled and resumes.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled``()`

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility``()`

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets``()`

List of the children *clove::Widget* instances.

`parentWidget``()`

The parent *clove::Widget*.

`name``()`

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName``(value)`

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled``()`

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled``(value)`

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass``()`

Custom css class(es).

`setStyleClass``(value)`

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style``()`

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(value)`

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.6 Class clove::Border

class *clove*::**Border** : **public** *clove*::*Widget*

A single-widget container framing the inner one with (a css-styled) border.

Public Functions

body()

The inner widget as widget configuration like for *clove::build()*.

setBody(value)

Setter for *body()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.7 Class clove::Button

`class clove::Button` : **`public`** *clove::Widget*

A button allows the user to trigger a particular program event.

Subclassed by *clove::PopupMenuButton*

Public Functions

`label``()`

The label text.

`setLabel``(value)`

Setter for *label()*.

Parameters

- `value`: The new value.

`icon``()`

The optional *clove::Icon* button icon.

`setIcon``(value)`

Setter for *icon()*.

Parameters

- `value`: The new value.

`checkable``()`

If the button is checkable (i.e. has a checked-flag).

If it has, the checked-flag is controlled by *checked()*.

`setCheckable``(value)`

Setter for *checkable()*.

Parameters

- `value`: The new value.

checked()

For a checkable button, return if it is checked.

See also *checkable()*.

User interactions do not toggle the checked flag by default. This must be scripted in own event handlers as needed.

setChecked(value)

Setter for *checked()*.

Parameters

- `value`: The new value.

OnClicked

Triggered when the user clicks on this button.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()

Sets the focus to this widget.

isAlive ()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth ()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass (clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()
The namescope this widget contains to.

nameScope ()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()
List of the children *clove::Widget* instances.

parentWidget ()
The parent *clove::Widget*.

name ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.8 Class clove::Carousel

`class clove::Carousel` : **`public`** *clove::TabView*

A carousel container.

Something like a tab bar, but automatically switching forward tabs in regular intervals (and with a different style).

Public Functions

`interval`()

The tab switching interval (in milliseconds).

`setInterval`(value)

Setter for *interval()*.

Parameters

- `value`: The new value.

`play`()

Begins automatical tab switching.

`pause`()

Stops automatic tab switching.

`isPlaying`()

If automatic tab switching is enabled.

next()
Switches to the next tab.

tabs()
The list of tabs.

Each tab is a widget configuration, like for *clove::build()*, with some additional optional keys which holds infos like the tab header text. So, for example, one item in that list could be `{view:'Something', ..., tabLabel:'Tab1'}`. See also *addTab()*.

setTabs(value)
Setter for *tabs()*.

Parameters

- `value`: The new value.

currentTab()
The index of the currently selected tab.

setCurrentTab(value)
Setter for *currentTab()*.

Parameters

- `value`: The new value.

userMayAddTabs()
If to show a button for adding new tabs.

If `true`, implement a handler for OnTabCreationRequested and create a tab with *addTab()* there.

setUserMayAddTabs(value)
Setter for *userMayAddTabs()*.

Parameters

- `value`: The new value.

tabBarLocation()
Where to place the tab bar.

Either `'top'`, `'left'`, `'bottom'` or `'right'`.

setTabBarLocation(value)
Setter for *tabBarLocation()*.

Parameters

- `value`: The new value.

switchInvisibleAnimationName()
Optional animation name for switching away from a tab.

setSwitchInvisibleAnimationName(value)
Setter for *switchInvisibleAnimationName()*.

Parameters

- `value`: The new value.

switchVisibleAnimationName()

Optional animation name for switching to a tab.

setSwitchVisibleAnimationName(value)

Setter for *switchVisibleAnimationName()*.

Parameters

- `value`: The new value.

switchInvisibleAnimationDuration()

Optional animation duration (in msec) for the switch away animation.

See also *clove::TabView::switchInvisibleAnimationName()*.

setSwitchInvisibleAnimationDuration(value)

Setter for *switchInvisibleAnimationDuration()*.

Parameters

- `value`: The new value.

switchVisibleAnimationDuration()

Optional animation duration (in msec) for the switch animation.

See also *clove::TabView::switchVisibleAnimationName()*.

setSwitchVisibleAnimationDuration(value)

Setter for *switchVisibleAnimationDuration()*.

Parameters

- `value`: The new value.

OnTabCreationRequested

Triggered when the user requested a new tab.

See also *userMayAddTabs()*.

This is a *clove.Event* instance. See Manual for details.

addTab(config)

Adds a new tab.

The configuration may contain the following additional keys:

- `tabLabel`: The tab header text.
- `tabIcon`: The tab icon as *clove::Icon*.
- `tabMayBeClosedByUser`: If the user may close this tab.

This method returns a *clove::Widget* which can be used for getting subwidgets or removing the tab.

Parameters

- `config`: A widget configuration like for *clove::build()*.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()

List of the children *clove::Widget* instances.

parentWidget ()

The parent *clove::Widget*.

name ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled ()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass ()

Custom css class(es).

setStyleClass (value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style ()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing``()`

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

`setStrictHorizontalSizing``(value)`

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing``()`

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing``(value)`

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch``()`

Alias for *verticalStretchAffinity()*.

`setVstretch``(value)`

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch``()`

Alias for *horizontalStretchAffinity()*.

`setHstretch``(value)`

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy``()`

If the widget is in busy state, typically resulting in a loading animation.

`setBusy``(value)`

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy``()`

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy``(token)`

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus``()`

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize``()`

Returns inner width and height of this widget.

`outerSize``()`

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.9 Class clove::CheckButton

class *clove*::**CheckButton** : **public** *clove*::*Widget*

A check button.

The user can choose to check and uncheck freely (in contrast to *clove::RadioButton*).

Public Functions

label()

The label text.

setLabel(value)

Setter for *label()*.

Parameters

- `value`: The new value.

checked()

If this check button is checked.

setChecked(value)

Setter for *checked()*.

Parameters

- `value`: The new value.

OnChanged

Triggered when this check button was selected.

This is a *clove.Event* instance. See Manual for details.

OnClicked

Triggered when this check button was selected. Same as OnChanged.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be ‘undefined’ for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing``()`

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing``(value)`

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch``()`

Alias for *verticalStretchAffinity()*.

`setVstretch``(value)`

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch``()`

Alias for *horizontalStretchAffinity()*.

`setHstretch``(value)`

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy``()`

If the widget is in busy state, typically resulting in a loading animation.

`setBusy``(value)`

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy``()`

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy``(token)`

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.10 Class clove::DataBinding

class *clove*::**DataBinding**
A data binding.

Read the Manual for details.

Public Functions

`DataBinding`(widget, propertyName, datasource, ptr, dataDirection, valueComparator, valueConverter)

Parameters

- `widget`: The *clove::Widget* to bind to the datasource.
- `propertyName`: The property name in the widget to bind.
- `datasource`: The *clove::Datasource* to bind to the widget.
- `ptr`: The *clove::DatasourceValuePointer*.
- `dataDirection`: See *clove::databind()*.
- `valueComparator`: Optional custom function which checks if two values are equal.
- `valueConverter`: Optional *clove::DataBindingConverter* for conversion between ui and data model.

`bind`()

Activates the binding.

`unbind`()

Deactivates the binding.

18.1.11 Class clove::DataBindingConverter

class *clove*::**DataBindingConverter**

A data binding converter maps values between ui representation and data model in a custom way.

Public Functions

`convertTo`(x)

Converts an original datasource value to a user interface representation.

Parameters

- `x`: The original datasource value.

`convertFrom`(y)

Converts a user interface representation to an original datasource value.

Parameters

- `y`: The user interface representation value.

18.1.12 Class clove::DataView

class *clove*::**DataView** : **public** *clove*::*Widget*

Base class for some views which shows hierarchical data from a *clove::Datasource*.

Read the Manual for details.

Although this is a fully-functional and implemented class, you should consider using one of the subclasses.

Subclassed by *clove::ListView*, *clove::TableView*, *clove::TreeView*

Public Functions

datasource()

The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

setDatasource(value)

Setter for *datasource()*.

Parameters

- `value`: The new value.

dataViewProcessor()

An optional `function(ptr, value)` for processing a datasource value to a display string.

setDataViewProcessor(value)

Setter for *dataViewProcessor()*.

Parameters

- `value`: The new value.

cellGenerator()

A generator function for complex cell content.

This method is called for each cell with `(clove::Widget, clove::DatasourceValuePointer)`. It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

setCellGenerator(value)

Setter for *cellGenerator()*.

Parameters

- `value`: The new value.

alwaysAllocateExpanderSpace()

If some space is allocated for an expander even for cells without children.

setAlwaysAllocateExpanderSpace(value)

Setter for *alwaysAllocateExpanderSpace()*.

Parameters

- `value`: The new value.

showOnlyFirstColumn()

If to show only the first column and hide the other ones.

setShowOnlyFirstColumn(value)

Setter for *showOnlyFirstColumn()*.

Parameters

- `value`: The new value.

hideExpanders()

If to hide all expanders.

setHideExpanders(value)

Setter for *hideExpanders()*.

Parameters

- `value`: The new value.

allowSelection()

If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

setAllowSelection(value)

Setter for *allowSelection()*.

Parameters

- `value`: The new value.

allowChecking()

If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

setAllowChecking(value)

Setter for *allowChecking()*.

Parameters

- `value`: The new value.

granularity()

The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

setGranularity(value)

Setter for *granularity()*.

Parameters

- `value`: The new value.

showChangeMenu()

If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:

- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

setShowChangeMenu(value)
Setter for *showChangeMenu()*.

Parameters

- `value`: The new value.

editOnGesture()
If the user shall be able to edit cells by double clicking/tapping them.

setEditOnGesture(value)
Setter for *editOnGesture()*.

Parameters

- `value`: The new value.

rowsResizable()
If the user shall be able to resize rows.

setRowsResizable(value)
Setter for *rowsResizable()*.

Parameters

- `value`: The new value.

columnsResizable()
If the user shall be able to resize columns.

setColumnsResizable(value)
Setter for *columnsResizable()*.

Parameters

- `value`: The new value.

selectCell(ptr)
Selects a cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

isCellSelected(ptr)
Checks if a given cell is selected.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

checkedCells()
Returns the checked cells as list of *clove::DatasourceValuePointer*.

setCheckedCells(ptrs)
Seturns the checked cells.

Parameters

- `ptrs`: A list of *clove::DatasourceValuePointer*.

isCellChecked(ptr)
Checks if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

setCellChecked(ptr, val)
Sets if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

selection()
Returns the selected item as *clove::DatasourceValuePointer*.

headersource()
The *clove::Headersource* providing row and column header configurations.

setHeadersource(value)
Setter for *headersource()*.

Parameters

- `value`: The new value.

gridVisible()
If to show a cell grid.

setGridVisible(value)
Setter for *gridVisible()*.

Parameters

- `value`: The new value.

nodeActivationNeedsDoubleClick()
If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

setNodeActivationNeedsDoubleClick(value)
Setter for *nodeActivationNeedsDoubleClick()*.

Parameters

- `value`: The new value.

expandCell(ptr)
Expands a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

expandCellRecursive(ptr)
Expands a given cell and all parents.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

collapseCell(ptr)
Collapses a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

isCellExpanded(ptr)
Checks if a given cell is expanded.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

editCell(ptr)
Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

OnSelectionChanged
Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be ‘undefined’ for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()`==0.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.13 Class clove::Datasource

class *clove*::**Datasource**
Base class for datasources.

Read the Manual for details.

Subclassed by *clove::AsyncDatasource*, *clove::NativeDatasource*, *clove::ProxyDatasource*

Public Functions

`getValue` (ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`getMetadata` (ptr)

Returns an object of metadata information (like icons, status infos used for styling, . . .). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`changeValue` (ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

`rowCount` (*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`columnCount` (*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`valuePointer` (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent` (ptr)

Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

valuePointerNavigateInDepth(ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

OnDataInsert

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataRemove

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataUpdate

Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

18.1.14 Class clove::DatasourceValuePointer

class *clove*::**DatasourceValuePointer**

A pointer to one node in a *clove::Datasource*.

Public Functions

DatasourceValuePointer(irow, icol, backendObject, datasource)

Typically called only inside a *clove::Datasource* implementation. From outside, you should use *clove::Datasource::valuePointer*.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `backendObject`: The backend object. This depends on the datasource implementation.
- `datasource`: The owning *clove::Datasource*.

irow

The row index.

icol

The column index.

datasource

The owning *clove::Datasource*.

backendObject

The backend object.

This depends on the datasource implementation.

parent ()

Returns the parent *clove::DatasourceValuePointer*.

sibling (irow, icol)

Returns a sibling *clove::DatasourceValuePointer*.

Parameters

- `irow`: The row index.
- `icol`: The column index.

rowCount ()

Returns the number of rows in this node.

columnCount ()

Returns the number of columns in this node.

getValue ()

Returns the value stored behind this pointer.

static **equals** (ptr1, ptr2)

Checks if two value pointers point to the same place.

Parameters

- `ptr1`: The first *clove::DatasourceValuePointer*.
- `ptr2`: The second *clove::DatasourceValuePointer*.

static **toPath** (ptr)

Returns an array-of-tuples representation for a value pointer.

See also *clove::DatasourceValuePointer::fromPath()*.

Parameters

- `ptr`: The *clove::DatasourceValuePointer* to convert.

static **fromPath** (datasource, path)

Returns a *clove::DatasourceValuePointer* for a array-of-tuples representation.

See also *clove::DatasourceValuePointer::toPath()*.

Parameters

- `datasource`: The *clove::Datasource* the value pointer shall correspond to.
- `path`: The array-of-tuples.

static **toString** (ptr)

Returns a string representation for a value pointer.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

18.1.15 Class clove::DateBox

`class clove::DateBox : public clove::EditBox`

A user input box for date input.

The actual behavior depends on the browser.

Public Functions

date()

The date value in the box as JavaScript Date.

setDate(value)

Setter for *date()*.

Parameters

- `value`: The new value.

readOnly()

If the edit box is read-only or editable by the user.

setReadOnly(value)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

text()

The current text.

setText(value)

Setter for *text()*.

Parameters

- `value`: The new value.

hintText()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

setHintText(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

autocompletionItems()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

setAutocompletionItems(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()
How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)
Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()
If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)
Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)
Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()
Returns the current text selection indices.

`OnChanged`
Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`
Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy (token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus ()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize ()

Returns inner width and height of this widget.

outerSize ()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.16 Class clove::Dialog

class *clove*::**Dialog** : **public** *clove*::*Widget*

A dialog is a container for a new sub user interface, decorated with a border and a title bar.

Public Functions

body()

The inner widget as widget configuration like for *clove::build()*.

setBody(value)

Setter for *body()*.

Parameters

- `value`: The new value.

title()

The dialog title text.

setTitle(value)

Setter for *title()*.

Parameters

- `value`: The new value.

titleicon()

The dialog title icon as *clove::Icon*.

setTitleicon(value)

Setter for *titleicon()*.

Parameters

- `value`: The new value.

close()

Closes and removes this dialog.

static show(dlg, showconfig)

Shows a dialog.

The root view of `dlg` is expected to be a *clove::Dialog* (or a subclass).

Parameters

- `dlg`: The widget configuration, as for *clove::build()*, which specifies the dialog.
- `showconfig`: A configuration object which specifies some presentation aspects.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.

- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize``()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize``()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout``()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus``()`

Sets the focus to this widget.

`isAlive``()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth`(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`getPreferredHeightForWidth`(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.17 Class clove::DropdownBox

class *clove*::**DropdownBox** : **public** *clove*::*EditComboBox*

A single-line text box (without edit functionality) with a popup list of values to select from.

Public Functions

popupItems()

A *clove.Datasource* with a list of items for the popup.

setPopupItems(value)

Setter for *popupItems()*.

Parameters

- `value`: The new value.

readOnly()

If the edit box is read-only or editable by the user.

setReadOnly(value)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

text()

The current text.

setText(value)

Setter for *text()*.

Parameters

- `value`: The new value.

hintText()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

setHintText(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

autocompletionItems()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

setAutocompletionItems(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter``()`

How to filter the autocompletion list.

Either `'startswith'`,`'contains'`,`function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter``(value)`

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText``()`

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText``(value)`

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection``(ifrom, ito)`

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection``()`

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty``(k, defaultV)`

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty``(k)`

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus``()`

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize``()`

Returns inner width and height of this widget.

`outerSize``()`

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.18 Class clove::EditBox

class `clove`**::EditBox** : **public** *clove::Widget*

A single-line box for text input from the user.

Subclassed by *clove::DateBox*, *clove::EditComboBox*, *clove::MultilineEditBox*, *clove::NumericEditBox*, *clove::PasswordEditBox*, *clove::TimeBox*

Public Functions

readOnly()

If the edit box is read-only or editable by the user.

setReadOnly(value)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

text()

The current text.

setText(value)

Setter for *text()*.

Parameters

- `value`: The new value.

hintText()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

setHintText(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

autocompletionItems()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

setAutocompletionItems(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

autocompletionFilter()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty(‘name’) `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *verticalStretchAffinity()*==0.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.19 Class clove::EditComboBox

class *clove*::**EditComboBox** : **public** *clove*::*EditBox*
A single-line box for text input from the user with an additional popup list of value recommendations.

Subclassed by *clove::DropdownBox*

Public Functions

`popupItems`()

A *clove.Datasource* with a list of items for the popup.

`setPopupItems`(value)

Setter for *popupItems()*.

Parameters

- `value`: The new value.

`readOnly`()

If the edit box is read-only or editable by the user.

`setReadOnly`(value)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

`text`()

The current text.

`setText`(value)

Setter for *text()*.

Parameters

- `value`: The new value.

`hintText`()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems`()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

setAutocompletionFilter(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

autocompletionOpenForNoText()

If to show the autocompletion list when the edit box is empty.

setAutocompletionOpenForNoText(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

setTextSelection(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

textSelection()

Returns the current text selection indices.

OnChanged

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

OnPopupTextSelected

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.20 Class clove::Event

class *clove*::**Event**

An event. It can have some handlers (or: listeners) and can be triggered.

Read the Manual for details about the Clove event mechanism.

Public Functions

Event ()

trigger (params)

Triggers the event.

This calls all registered handlers.

Parameters

- `params`: Additional information about the event incidence.

addHandler (fct, owner)

Adds a handler (or: listener) to this event.

Whenever the event is triggered afterwards, the new handler is called.

Parameters

- `fct`: A handler function.
- `owner`: Optional owner widget. If given, removeHandler will be called automatically after widget removal.

hasHandlers ()

Checks if this event has any handlers (otherwise triggering it doesn't have an effect).

addHandlerOnce (fct, owner)

Like *addHandler()*, but automatically removes the handler after the event occurred once.

Parameters

- `fct`: A handler function.
- `owner`: Optional owner widget. If given, removeHandler will be called automatically after widget removal.

removeHandler (fct)

Removes a handler from this event.

Parameters

- `fct`: The handler function to remove.

18.1.21 Class clove::EventArgs

class *clove*::**EventArgs**

Arguments for a particular incidence of a *clove::Event*.

It carries some additional informations about it. The details depend on the event (and the component which triggers it).

Public Functions

`EventArgs`(params)

Parameters

- `params`: Additional information about the event incidence.

`skipExecution`()

Marks the further event handling for skipping.

18.1.22 Class clove::Expander

`class` *`clove`*::**`Expander`** : **`public`** *clove*::*Widget*

A single-widget container which can be expanded or collapsed.

Public Functions

`isExpanded`()

If the expander is expanded.

`setIsExpanded`(value)

Setter for *isExpanded()*.

Parameters

- `value`: The new value.

`collapsedIcon`()

The icon for the expander when collapsed.

`setCollapsedIcon`(value)

Setter for *collapsedIcon()*.

Parameters

- `value`: The new value.

`collapsedLabel`()

The label for the expander when collapsed.

`setCollapsedLabel`(value)

Setter for *collapsedLabel()*.

Parameters

- `value`: The new value.

`expandedIcon`()

The icon for the expander when expanded.

`setExpandedIcon`(value)

Setter for *expandedIcon()*.

Parameters

- `value`: The new value.

expandedLabel()
The label for the expander when expanded.

setExpandedLabel(value)
Setter for *expandedLabel()*.

Parameters

- `value`: The new value.

body()
The inner *clove::Widget*.

setBody(value)
Setter for *body()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)
General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.

  - `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()
Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()
Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled``()`

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled``(value)`

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass``()`

Custom css class(es).

`setStyleClass``(value)`

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style``()`

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(value)`

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility``()`

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility``(value)`

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing``()`

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing``(value)`

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus``()`

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch ()
Alias for *horizontalStretchAffinity()*.

setHstretch (value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy ()
If the widget is in busy state, typically resulting in a loading animation.

setBusy (value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy ()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy (token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus ()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize ()
Returns inner width and height of this widget.

outerSize ()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.23 Class clove::FilterProxyDatasource

class `clove`::**`FilterProxyDatasource`** : **public** *clove*::*ProxyDatasource*

A *clove::Datasource* implementation which filters another *clove::Datasource*.

Public Functions

`FilterProxyDatasource`(datasource, rowfilterfct, colfilterfct)

Parameters

- `datasource`: The source *clove::Datasource*.
- `rowfilterfct`: A `function(datasource, idx, parent)` returning `true` if the filter accepts this row (optional).
- `colfilterfct`: A `function(datasource, idx, parent)` returning `true` if the filter accepts this column (optional).

`setRowFilter`(rowfilterfct)

Sets the row filter function.

Parameters

- `rowfilterfct`: The row filter function. See constructor for details.

`setColumnFilter`(colfilterfct)

Sets the column filter function.

Parameters

- `colfilterfct`: The column filter function. See constructor for details.

`setDatasource`(datasource)

Sets the source datasource.

Parameters

- `datasource`: The source *clove::Datasource*.

`proxyPointerToNativePointer`(proxyptr)
Translates a *clove::DatasourceValuePointer* from this proxy datasource to a one for the source datasource.

Parameters

- `proxyptr`: A value pointer.

`nativePointerToProxyPointer`(nativeptr)
Translates a *clove::DatasourceValuePointer* from the source datasource to a one for this proxy datasource.

Parameters

- `nativeptr`: A value pointer.

`refresh`()
Refreshes the filtering.

Call this when the outer conditions changed and the filters must be applied again.

`getValue`(ptr)
Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`getMetadata`(ptr)
Returns an object of metadata information (like icons, status infos used for styling, . . .). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`changeValue`(ptr, value)
Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

`rowCount`(*parent*)
Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

`columnCount`(*parent*)
Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer (irow, icol, *parent*)
Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

parent (ptr)
Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

valuePointerNavigateInDepth (ptr, direction, mayexpandfct)
Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

OnDataInsert
Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataRemove
Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataUpdate
Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

getRowHeader (irow, *parent*)
Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

getColumnHeader (irow, *parent*)
Returns the header configuration for a given column.

Parameters

- `irow`: The column index.

- `parent`: The parent node as *clove::DatasourceValuePointer*.

rowHeadersVisible(*parent*)
If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

columnHeadersVisible(*parent*)
If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

OnHeaderDataInsert
Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataRemove
Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataUpdate
Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

OnHeaderVisibilityUpdated
Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

18.1.24 Class clove::FlatLayout

class *clove*::**FlatLayout** : **public** *clove*::*Layout*
A mixin for flat layout implementations.

This layout shows one of its childs in full sizes and hides all the others.

Subclassed by *clove::FlatView*

Public Functions

switchInvisibleAnimationName()
Optional animation name for disabling a view.

setSwitchInvisibleAnimationName(value)
Setter for *switchInvisibleAnimationName()*.

Parameters

- `value`: The new value.

switchVisibleAnimationName()
Optional animation name for enabling a view.

setSwitchVisibleAnimationName(value)
Setter for *switchVisibleAnimationName()*.

Parameters

- `value`: The new value.

switchInvisibleAnimationDuration()
Optional animation duration (in msec) for the disabling animation.

See also *clove::FlatLayout::switchInvisibleAnimationName()*.

setSwitchInvisibleAnimationDuration(value)
Setter for *switchInvisibleAnimationDuration()*.

Parameters

- `value`: The new value.

switchVisibleAnimationDuration()
Optional animation duration (in msec) for the enabling animation.

See also *clove::FlatLayout::switchVisibleAnimationName()*.

setSwitchVisibleAnimationDuration(value)
Setter for *switchVisibleAnimationDuration()*.

Parameters

- `value`: The new value.

collapseHidden()
If the non-visible widgets shall be collapsed (affects layouting).

setCollapseHidden(value)
Setter for *collapseHidden()*.

Parameters

- `value`: The new value.

currentView()
The index (integer) of the currently visible child.

setCurrentView(value)
Setter for *currentView()*.

Parameters

- `value`: The new value.

children()
List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)
Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)
Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()
Removes all childs from the layout.

18.1.25 Class clove::FlatView

class *clove*::**FlatView** : **public** *clove*::*Widget*, **public** *clove*::*FlatLayout*
A container which shows one of its child widgets in full size and hides all others.

Public Functions

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)
General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.

- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()
Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()
Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`switchInvisibleAnimationName`()

Optional animation name for disabling a view.

`setSwitchInvisibleAnimationName`(value)

Setter for *switchInvisibleAnimationName()*.

Parameters

- `value`: The new value.

`switchVisibleAnimationName`()

Optional animation name for enabling a view.

`setSwitchVisibleAnimationName`(value)

Setter for *switchVisibleAnimationName()*.

Parameters

- `value`: The new value.

`switchInvisibleAnimationDuration`()

Optional animation duration (in msec) for the disabling animation.

See also *clove::FlatLayout::switchInvisibleAnimationName()*.

`setSwitchInvisibleAnimationDuration`(value)

Setter for *switchInvisibleAnimationDuration()*.

Parameters

- `value`: The new value.

`switchVisibleAnimationDuration`()

Optional animation duration (in msec) for the enabling animation.

See also *clove::FlatLayout::switchVisibleAnimationName()*.

`setSwitchVisibleAnimationDuration`(value)

Setter for *switchVisibleAnimationDuration()*.

Parameters

- `value`: The new value.

collapseHidden()

If the non-visible widgets shall be collapsed (affects layouting).

setCollapseHidden(value)

Setter for *collapseHidden()*.

Parameters

- `value`: The new value.

currentView()

The index (integer) of the currently visible child.

setCurrentView(value)

Setter for *currentView()*.

Parameters

- `value`: The new value.

children()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)

Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()

Removes all childs from the layout.

18.1.26 Class clove::Form

class *clove*::**Form** : **public** *clove*::*Widget*

A container for a form-like with a label for each entry in a row-oriented alignment.

Public Functions

`sections`()

A list of widget configurations as for *clove::build()*. Each defines one section in the form.

Define the labels in the `formSectionLabel` property of each widget.

`setSections`(value)

Setter for *sections()*.

Parameters

- `value`: The new value.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()
Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()
Sets the focus to this widget.

isAlive ()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth ()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth ()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.27 Class clove::Grid

`class clove::Grid` : **`public`** *clove*::*Widget*, **`public`** *clove*::*GridLayout*

A container for aligning child widgets in a grid.

Set the `row` and `col` property in the child widget configurations to some numbers for assigning them to cells.

Public Functions

`declareProperty``(k, defaultV)`

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty``(k)`

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()
Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)
Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)
Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()
The namescope this widget contains to.

`nameScope`()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`insertRow`(i)
Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

`insertColumn`(i)
Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

`removeRow`(i)
Removes a row from the grid.

Parameters

- `i`: The row index.

`removeColumn`(i)
Removes a column from the grid.

Parameters

- `i`: The column index.

children()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)

Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()

Removes all childs from the layout.

18.1.28 Class clove::GridLayout

class *clove*::**GridLayout** : **public** *clove*::*Layout*

A mixin for grid layout implementations.

Subclassed by *clove::Grid*, *clove::StackLayout*

Public Functions

insertRow(i)

Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

insertColumn(i)

Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

removeRow(i)

Removes a row from the grid.

Parameters

- `i`: The row index.

removeColumn(i)

Removes a column from the grid.

Parameters

- `i`: The column index.

children()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)

Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()

Removes all childs from the layout.

18.1.29 Class clove::Headersource

class *clove*::**Headersource**

Base class for headersources.

A headersource provides information for headers in data views. Read the Manual for details.

Subclassed by *clove::AjaxAsyncDatasource*, *clove::AsyncDatasource*, *clove::NativeDatasource*, *clove::ProxyDatasource*

Public Functions

getRowHeader(irow, parent)

Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

getColumnHeader(irow, parent)

Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

rowHeadersVisible(parent)

If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

columnHeadersVisible(parent)
If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

OnHeaderDataInsert
Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataRemove
Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataUpdate
Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

OnHeaderVisibilityUpdated
Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

18.1.30 Class clove::HorizontalStack

class *clove*::**HorizontalStack** : **public** *clove*::*Widget*, **public** *clove*::*StackLayout*
A container which stacks child widgets column-wise.

Public Functions

cols()
The list of child widgets as widget configurations like for *clove::build()*.

setCols(value)
Setter for *cols()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`insertRow`(i)
Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

insertColumn(i)
Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

removeRow(i)
Removes a row from the grid.

Parameters

- `i`: The row index.

removeColumn(i)
Removes a column from the grid.

Parameters

- `i`: The column index.

children()
List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)
Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)
Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()
Removes all childs from the layout.

18.1.31 Class clove::HtmlView

class *clove*::**HtmlView** : **public** *clove*::*Widget*
A host for arbitrary html content.

Public Functions

`contentRoot` ()

The root dom node of the html content.

`setContentRoot` (value)

Setter for *contentRoot()*.

Parameters

- `value`: The new value.

`declareProperty` (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty` (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty` (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty` (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init` (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()
The namescope this widget contains to.

nameScope ()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()
List of the children *clove::Widget* instances.

parentWidget ()
The parent *clove::Widget*.

name ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled ()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.32 Class clove::I18N

`class` *`clove`*`::`**`I18N`**
Internationalization support class.

Always use the instance *clove.i18n*. Read the Manual for details.

Public Functions

`addString`(id, texts)
Adds a text to the internationalization text table.

The keys are ISO 639-1 language codes.

Example: `addString('PleaseWait', {'de':'Bitte warten', 'en':'Please wait'})`

Parameters

- `id`: The text identifier name, like 'FooBar'.
- `texts`: A configuration object which holds a text representation for all target languages.

`setLanguage`(langcode)
Sets the current language.

You don't need to call this function, unless you want to override the user's language as it is configured in the operating system or browser.

Parameters

- `langcode`: The new current language (as ISO 639-1 language code).

`static` **`parseLangCode`**(lang)
Returns a clove string representation.

Only indended to be used by the infrastructure.

Parameters

- `lang`: A language code string as understood by *addString()*.

18.1.33 Class clove::Icon

class *clove*::**Icon**

An icon.

Can be shown at many places, like in buttons, menus or *clove::IconView*.

Construct instances with *clove::Icon::byUrl()* or *clove::Icon::bySymbol()*.

Public Functions

static byUrl(url)

Creates a *clove::Icon* by a url pointing to an image.

Parameters

- `url`: The image url.

static bySymbol(chr)

Creates a *clove::Icon* by a text character.

Parameters

- `chr`: The text character.

createHtml(height)

Creates an html representation.

This method is used by widget implementations.

Parameters

- `height`: The icon height in pixels.

18.1.34 Class clove::IconView

class *clove*::**IconView** : **public** *clove*::*Widget*

Shows an icon.

The icon can come from an image file or from a Unicode symbol.

Public Functions

size()

The icon size as css length.

setSize(value)

Setter for *size()*.

Parameters

- `value`: The new value.

icon()

The *clove::Icon* to show.

setIcon (value)

Setter for *icon()*.

Parameters

- `value`: The new value.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()
Custom css class(es).

`setStyleClass`(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)
Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.35 Class clove::ImageView

`class` `clove`**`::ImageView`** : **`public`** *clove*::*Widget*

Shows an image.

This does not contain any viewer controls like zoom buttons.

Public Functions

`source``()`

The image source URL.

`setSource``(value)`

Setter for *source()*.

Parameters

- `value`: The new value.

`zoom``()`

The zoom level.

1.0 is normal, larger values zoom in, smaller values zoom out. Specify ‘auto’ for automatically adjusting it to the available room.

`setZoom``(value)`

Setter for *zoom()*.

Parameters

- `value`: The new value.

`keepAspectRatio``()`

For ‘auto’ zoom: Whether to keep aspect ratio of the image.

`setKeepAspectRatio``(value)`

Setter for *keepAspectRatio()*.

Parameters

- `value`: The new value.

OnLoaded

Triggered when the image loading ended.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(`w`)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`getPreferredHeightForWidth``(w)`

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass``(clss)`

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass``(clss)`

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass``(clss)`

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass``()`
Custom css class(es).

`setStyleClass``(value)`
Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style``()`
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(value)`
Setter for *style()*.

Parameters

- `value`: The new value.

`visibility``()`
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility``(value)`
Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing``()`
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing``(value)`
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus``()`
If the widget can have the keyboard focus.

`setMayFocus``(value)`
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity``()`
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.36 Class clove::Label

`class clove::Label : public clove::Widget`

A text label.

Public Functions

`label()`

The label text.

For rich content, use *htmlContent()* instead.

`setLabel(value)`

Setter for *label()*.

Parameters

- `value`: The new value.

`htmlContent()`

The label content as html.

For plain text, use *label()* instead.

`setHtmlContent(value)`

Setter for *htmlContent()*.

Parameters

- `value`: The new value.

`focusBuddy()`

A *clove::Widget* or a widget name for a focus buddy.

Whenever the user clicks on this label, it will focus the buddy.

`setFocusBuddy(value)`

Setter for *focusBuddy()*.

Parameters

- `value`: The new value.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope()`
The namescope this widget contains to.

`nameScope()`
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove(removeconfig)`
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled()`
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility()`
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets()`
List of the children *clove::Widget* instances.

`parentWidget()`
The parent *clove::Widget*.

`name()`
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName(value)`
Setter for *name()*.

Parameters

- `value`: The new value.

`enabled()`
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled(value)`
Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass()`
Custom css class(es).

`setStyleClass(value)`
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.37 Class clove::Layout

`class clove::Layout`
A mixin for a widget classes which align child widgets in some way.

This is a base type and has some more interesting sub-classes.

Subclassed by *clove::FlatLayout*, *clove::GridLayout*, *clove::WrapLayout*

Public Functions

`children``()`
List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren``(value)`
Setter for *children()*.

Parameters

- `value`: The new value.

`addChild``(value)`
Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds``()`
Removes all childs from the layout.

18.1.38 Class clove::ListView

`class clove::ListView` : **`public`** *clove::DataView*
A view which shows a *clove::Datasource* in a list.

Public Functions

`datasource``()`
The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

`setDatasource``(value)`
Setter for *datasource()*.

Parameters

- `value`: The new value.

dataViewProcessor()
An optional `function(ptr, value)` for processing a datasource value to a display string.

setDataViewProcessor(value)
Setter for *dataViewProcessor()*.

Parameters

- `value`: The new value.

cellGenerator()
A generator function for complex cell content.

This method is called for each cell with (`clove::Widget`, `clove::DatasourceValuePointer`). It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

setCellGenerator(value)
Setter for *cellGenerator()*.

Parameters

- `value`: The new value.

alwaysAllocateExpanderSpace()
If some space is allocated for an expander even for cells without children.

setAlwaysAllocateExpanderSpace(value)
Setter for *alwaysAllocateExpanderSpace()*.

Parameters

- `value`: The new value.

showOnlyFirstColumn()
If to show only the first column and hide the other ones.

setShowOnlyFirstColumn(value)
Setter for *showOnlyFirstColumn()*.

Parameters

- `value`: The new value.

hideExpanders()
If to hide all expanders.

setHideExpanders(value)
Setter for *hideExpanders()*.

Parameters

- `value`: The new value.

allowSelection()
If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

setAllowSelection(value)

Setter for *allowSelection()*.

Parameters

- `value`: The new value.

allowChecking()

If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

setAllowChecking(value)

Setter for *allowChecking()*.

Parameters

- `value`: The new value.

granularity()

The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

setGranularity(value)

Setter for *granularity()*.

Parameters

- `value`: The new value.

showChangeMenu()

If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:

- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

setShowChangeMenu(value)

Setter for *showChangeMenu()*.

Parameters

- `value`: The new value.

editOnGesture()

If the user shall be able to edit cells by double clicking/tapping them.

setEditOnGesture(value)

Setter for *editOnGesture()*.

Parameters

- `value`: The new value.

rowsResizable()

If the user shall be able to resize rows.

setRowsResizable(value)

Setter for *rowsResizable()*.

Parameters

- `value`: The new value.

columnsResizable()

If the user shall be able to resize columns.

setColumnsResizable(value)

Setter for *columnsResizable()*.

Parameters

- `value`: The new value.

selectCell(ptr)

Selects a cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

isCellSelected(ptr)

Checks if a given cell is selected.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

checkedCells()

Returns the checked cells as list of *clove::DatasourceValuePointer*.

setCheckedCells(ptrs)

Seturns the checked cells.

Parameters

- `ptrs`: A list of *clove::DatasourceValuePointer*.

isCellChecked(ptr)

Checks if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

setCellChecked(ptr, val)

Sets if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

selection()
Returns the selected item as *clove::DatasourceValuePointer*.

headersource()
The *clove::Headersource* providing row and column header configurations.

setHeadersource(value)
Setter for *headersource()*.

Parameters

- `value`: The new value.

gridVisible()
If to show a cell grid.

setGridVisible(value)
Setter for *gridVisible()*.

Parameters

- `value`: The new value.

nodeActivationNeedsDoubleClick()
If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

setNodeActivationNeedsDoubleClick(value)
Setter for *nodeActivationNeedsDoubleClick()*.

Parameters

- `value`: The new value.

expandCell(ptr)
Expands a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

expandCellRecursive(ptr)
Expands a given cell and all parents.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

collapseCell(ptr)
Collapses a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

isCellExpanded(ptr)
Checks if a given cell is expanded.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

editCell (ptr)

Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

OnSelectionChanged

Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()

Sets the focus to this widget.

`isAlive`()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth`()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth`()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled``()`

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled``(value)`

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass``()`

Custom css class(es).

`setStyleClass``(value)`

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style``()`

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(value)`

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility``()`

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility``(value)`

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing``()`

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing``(value)`

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus``()`

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.39 Class clove::MainView

`class clove::MainView : public` *clove::Widget*
A typical main view, including a *clove::Toolbar* and a container for an inner body widget.

Public Functions

`sidebar``()`
The sidebar widget as widget configuration like for *clove::build()*.

`setSidebar``(value)`
Setter for *sidebar()*.

Parameters

- `value`: The new value.

`bodyLeft``()`
The left inner widget as widget configuration like for *clove::build()*.

`setBodyLeft``(value)`
Setter for *bodyLeft()*.

Parameters

- `value`: The new value.

`bodyRight``()`
The right inner widget as widget configuration like for *clove::build()*.

`setBodyRight``(value)`
Setter for *bodyRight()*.

Parameters

- `value`: The new value.

bodyLeftViewActionLabel()
The title string for the left inner widget.

setBodyLeftViewActionLabel(value)
Setter for *bodyLeftViewActionLabel()*.

Parameters

- `value`: The new value.

bodyRightViewActionLabel()
The title string for the right inner widget.

setBodyRightViewActionLabel(value)
Setter for *bodyRightViewActionLabel()*.

Parameters

- `value`: The new value.

splitterPosition()
The position of the splitter between the left and right inner widget as number between `0.0` and `1.0`.

setSplitterPosition(value)
Setter for *splitterPosition()*.

Parameters

- `value`: The new value.

switchToRightBody()
If in small mode, it switches the view to the right body.

switchToLeftBody()
If in small mode, it switches the view to the left body.

head1()
The 1st header text.

setHead1(value)
Setter for *head1()*.

Parameters

- `value`: The new value.

head2()
The 2nd header text.

setHead2(value)
Setter for *head2()*.

Parameters

- `value`: The new value.

headControl()
An optional widget for arbitrary control purposes as widget configuration like for *clove::build()*. It's displayed below the menu bar in full width.

setHeadControl(value)
Setter for *headControl()*.

Parameters

- `value`: The new value.

headControlWidget()
Returns the control widget as *clove::Widget* (or undefined if no *headControl()* is set).

icon()
The header icon as *clove::Icon*.

setIcon(value)
Setter for *icon()*.

Parameters

- `value`: The new value.

actions()
Menu actions.

See *clove::Menubar::actions()*.

setActions(value)
Setter for *actions()*.

Parameters

- `value`: The new value.

showOnly()
Specifies if to forcefully show just one of the two main panels (`0`, `1`, `undefined`).

setShowOnly(value)
Setter for *showOnly()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`getPreferredHeightForWidth`(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.40 Class clove::MediaPlayer

`class` *`clove`*`::`**`MediaPlayer`** : `public` *clove*::*Widget*

A base class for media player widgets.

This is an abstract class. Use one of the subclasses.

Note: This interface provides actions and events for typical usage. For other scenarios, use *nativeNode()* for retrieving the `<audio>` or `<video>` html dom node. They provide a more powerful api.

Subclassed by *clove::AudioPlayer*, *clove::VideoPlayer*

Public Functions

`autoPlay``()`

If to automatically start playback as soon as possible.

`setAutoPlay``(value)`

Setter for *autoPlay()*.

Parameters

- `value`: The new value.

`showControls``()`

If to show user controls for play, pause and more.

`setShowControls``(value)`

Setter for *showControls()*.

Parameters

- `value`: The new value.

`currentMediaPosition``()`

The current media playback position in seconds.

`setCurrentMediaPosition``(value)`

Setter for *currentMediaPosition()*.

Parameters

- `value`: The new value.

`loop``()`

If to playback in an endless loop.

`setLoop``(value)`

Setter for *loop()*.

Parameters

- `value`: The new value.

`muted``()`

If to mute the audio playback.

`setMuted``(value)`

Setter for *muted()*.

Parameters

- `value`: The new value.

`preload`()

If to begin loading the media data as soon as possible.

`setPreload`(value)

Setter for *preload()*.

Parameters

- `value`: The new value.

`volume`()

The audio playback volume between `0.0` and `1.0`.

`setVolume`(value)

Setter for *volume()*.

Parameters

- `value`: The new value.

`source`()

The media source URL.

`setSource`(value)

Setter for *source()*.

Parameters

- `value`: The new value.

`duration`()

The duration of the loaded media source in seconds.

`hasEnded`()

If the playback has ended (i.e. reached the end).

`isPaused`()

If the playback is logically paused.

This does not return `true` just when loading stalls.

`nativeNode`()

Returns the native html dom node.

`play`()

Starts playback.

`pause`()

Pauses playback.

`OnLoadingAborted`

Triggered when the media loading aborted for some reasons (e.g. network errors).

This is a *clove.Event* instance. See Manual for details.

OnReadyForPlayback

Triggered when there are enough data for starting playback.

This is a *clove.Event* instance. See Manual for details.

OnDurationChanged

Triggered when the playback duration changed.

See also *duration()*.

This is a *clove.Event* instance. See Manual for details.

OnHasEnded

Triggered when the playback has ended.

See also *hasEnded()*.

This is a *clove.Event* instance. See Manual for details.

OnIsPaused

Triggered when the playback logically paused.

This is not triggered when loading stalls.

This is a *clove.Event* instance. See Manual for details.

OnIsPlaying

Triggered when the playback logically starts.

This is not triggered when loading has stalled and resumes.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)
Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()
Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()
Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()
Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()
Sets the focus to this widget.

`isAlive``()`
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth``()`
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth``()`
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth``(w)`
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`computePreferredHeightForWidth``(w)`
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

`getMinimalWidth``()`
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getPreferredWidth``()`
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

`getMinimalHeightForWidth``(w)`
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`getPreferredHeightForWidth``(w)`
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass``(clss)`
Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass (clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass (clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()

The namescope this widget contains to.

nameScope ()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()

List of the children *clove::Widget* instances.

parentWidget ()

The parent *clove::Widget*.

name ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled ()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass ()

Custom css class(es).

setStyleClass (value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style ()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle (value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility ()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility (value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing ()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don’t trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.41 Class clove::Menubar

class *clove*::**Menubar** : **public** *clove::AbstractMenu*
A menu bar.

Public Functions

actions()
The actions to be shown in this menu.

It is a list of action configuration objects, each having a structure like `{name:’myaction’, label:’My menu action’}. It can have a list of sub-items in the subactions` property.

Instead of a simple list, it may also be a *clove::Datasource* with the action configuration structures aligned in rows.

Use *clove::MenuSeparator* for a separator.

One action allows this properties:

- `name`: The action name.
- `label`: The label string.
- `icon`: A *clove::Icon*.
- `disabled`: If it is disabled.

- `invisible`: If it is invisible.
- `checkable`: If it is checkable.
- `checked`: If it is checked.

setActions (value)
Setter for *actions()*.

Parameters

- `value`: The new value.

OnActionTriggered
Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

OnBeforeSubactionsExpanded
Triggered just before a branch of subaction is expanded.

Implement this event for populating it dynamically.

This is a *clove.Event* instance. See Manual for details.

declareProperty (k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)
General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()

Sets the focus to this widget.

isAlive ()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth ()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus``()`
If the widget can have the keyboard focus.

`setMayFocus``(value)`
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity``()`
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity``(value)`
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity``()`
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity``(value)`
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing``()`
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing``(value)`
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing``()`
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing``(value)`
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch``()`
Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.42 Class clove::ModalPanel

`class` `clove`**`::ModalPanel`** : **`public`** *clove*::*Widget*

A surface for implementing modality.

For a certain time, it will be inserted before the main user interface (i.e. higher on z-axis) in order to block user interaction with it.

It is typically not required to use it directly. Try *clove::Dialog::show()* and *clove::utils::popup()* before.

Public Functions

`fullyTransparent`()

If the modal panel is fully transparent (i.e. not really visible) instead of semi-transparent.

`setFullyTransparent`(value)

Setter for *fullyTransparent()*.

Parameters

- `value`: The new value.

`OnClicked`

Triggered when the user clicks on this modal panel.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()
Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()
Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()
Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()

List of the children *clove::Widget* instances.

parentWidget ()

The parent *clove::Widget*.

name ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled ()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass ()

Custom css class(es).

setStyleClass (value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style ()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing``()`

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

`setStrictHorizontalSizing``(value)`

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing``()`

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing``(value)`

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch``()`

Alias for *verticalStretchAffinity()*.

`setVstretch``(value)`

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch``()`

Alias for *horizontalStretchAffinity()*.

`setHstretch``(value)`

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy``()`

If the widget is in busy state, typically resulting in a loading animation.

`setBusy``(value)`

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.43 Class clove::MultilineEditBox

class *clove*::**MultilineEditBox** : **public** *clove*::*EditBox*

A multi-line box for text input from the user.

Public Functions

readOnly()

If the edit box is read-only or editable by the user.

setReadOnly(value)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

text()

The current text.

setText(value)

Setter for *text()*.

Parameters

- `value`: The new value.

hintText()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

setHintText(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

autocompletionItems()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

setAutocompletionItems(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

autocompletionFilter()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

setAutocompletionFilter(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `andx.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.44 Class clove::NameScope

`class` *`clove`*`::`**`NameScope`**

A mixin realizing a new namescope.

Read the Manual for details.

Subclassed by *clove::RootNameScope*

Public Functions

`getByName` (name)

Finds a widget by name within this namescope.

Parameters

- `name`: The widget name.

`addChildNameScope` (childnamescope)

Adds a namescope to the children of this one, so a searcher will also search in this child.

Parameters

- `childnamescope`: The child namescope.

18.1.45 Class clove::NativeDatasource

class *clove*::**NativeDatasource** : **public** *clove*::*Datasource*, **public** *clove*::*Headersource*

A *clove::Datasource* implementation which stores all data in-memory.

It provides an interface for populating it with data and can directly be constructed and used. It also implements *clove::Headersource*, so it can also carry row and column header configurations.

Public Functions

NativeDatasource(config)

The configuration object allows the following keys:

- `readonly`: If this datasource may be changed from outside.
- `rowHeadersVisible`: If row headers shall be visible in presentation.
- `columnHeadersVisible`: If column headers shall be visible in presentation.

  Parameters

  - `config`: A configuration object.

root()

The root node as clove::NativeDatasourceValue.

valuePointerToNativeNode(ptr)

Returns the clove::NativeDatasourceValue for a node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

insertRow(ir, *parent*)

Inserts a new empty row.

Parameters

- `ir`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

insertColumn(ic, *parent*)

Inserts a new empty column.

Parameters

- `ic`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

appendRow(*parent*)

Appends a new empty row (like inserting to the end).

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

appendColumn (*parent*)

Appends a new empty column (like inserting to the end).

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

removeRow (ir, *parent*)

Removes a row.

Parameters

- `ir`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

removeColumn (ic, *parent*)

Removes a column.

Parameters

- `ic`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

setValue (ptr, value)

Sets a new value to a node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new node value.

setRootValue (value)

Sets a new value to the root node.

This is just a shorter variant of `setValue(undefined, value)` (for scalar usage).

Parameters

- `value`: The new node value.

setMetadata (ptr, key, value)

Sets a metadata key for a node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `key`: The metadata key, like ‘icon’.
- `value`: The new value.

setRowHeader (irow, *parent*, headercfg)

Sets a row header configuration.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.
- `headercfg`: The header configuration.

setColumnHeader(icol, *parent*, headercfg)
Sets a column header configuration.

Parameters

- `icol`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.
- `headercfg`: The header configuration.

getValue(ptr)
Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata(ptr)
Returns an object of metadata information (like icons, status infos used for styling, …). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

changeValue(ptr, value)
Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount(*parent*)
Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount(*parent*)
Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer(irow, icol, *parent*)
Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent` (ptr)
Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

`valuePointerNavigateInDepth` (ptr, direction, mayexpandfct)
Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

`OnDataInsert`
Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`
Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`
Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)
Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader` (irow, *parent*)
Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible` (*parent*)
If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible`(*parent*)

If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

18.1.46 Class clove::NativeDatasourceNode

`class clove::NativeDatasourceNode`

A node value implementation for *clove::NativeDatasource*.

Not intended for direct construction!

Public Functions

`rowCount`()

Returns the number of children rows.

`columnCount`()

Returns the number of children columns.

`toRow`(ir)

Returns the *clove::NativeDatasourceNode* which is in the same column as this one, but in a different row.

Parameters

- `ir`: The row index.

`toColumn`(ic)

Returns the *clove::NativeDatasourceNode* which is in the same row as this one, but in a different column.

Parameters

- `ic`: The column index.

valuePointer()
Returns a *clove::DatasourceValuePointer* pointing to this node.

getValue()
Returns the node value.

getMetadata()
Returns the node metadata.

See also *clove::Datasource::getMetadata()*.

setValue(v)
Sets the node value.

Parameters

- `v`: The new value.

setMetadata(k, v)
Sets a node metadata value.

Parameters

- `k`: The metadata key, like 'icon'.
- `v`: The new value.

removeRow(ir)
Removes a row in this node.

Parameters

- `ir`: The row index.

removeColumn(ic)
Removes a column in this node.

Parameters

- `ic`: The column index.

addRow(vals)
Adds a row to this node.

Parameters

- `vals`: A list of new values (one for each column).

addColumn(vals)
Adds a column to this node.

Parameters

- `vals`: A list of new values (one for each row).

insertRow(i, vals)
Inserts a row to this node.

Parameters

- `i`: The insertion position.
- `vals`: A list of new values (one for each column).

insertColumn (i, vals)

Inserts a column to this node.

Parameters

- `i`: The insertion position.
- `vals`: A list of new values (one for each row).

getChild (irow, icol)

Returns a child *clove::NativeDatasourceNode*.

Parameters

- `irow`: The row index.
- `icol`: The column index.

18.1.47 Class clove::NotificationController

class *clove*::**NotificationController**

Controller for clove notifications.

Notifications are small popups, typically in the top/right corner of the window. They can inform the user about some status changes or provide some ways for user interactions.

Use *clove::notifications*.

Public Functions

notify (config, notificationConfig)

Opens a new notification.

notificationConfig may contain:

- `timeout`: Closes the notification automatically after this amount of seconds.
- `closeable`: If to provide a close button.
- `priority`: Controls stuff like ordering. Default is 0, higher values mean more importance.

Return The *clove::Widget* implementing the notification. Use it for operations like removal or getting subwidget. This widget spans a new *clove::NameScope*!

Parameters

- `config`: A widget configuration for the notification body, as for *clove::build()*.
- `notificationConfig`: Some configuration aspects about the notification.

18.1.48 Class clove::NumericEditBox

class *clove*::**NumericEditBox** : **public** *clove*::*EditBox*

A text box with up/down buttons for numbers.

Public Functions

min()

The minimum value.

setMin(*value*)

Setter for *min()*.

Parameters

- `value`: The new value.

max()

The maximum value.

setMax(*value*)

Setter for *max()*.

Parameters

- `value`: The new value.

stepsize()

The step size.

setStepsize(*value*)

Setter for *stepsize()*.

Parameters

- `value`: The new value.

value()

The current numeric value.

setValue(*value*)

Setter for *value()*.

Parameters

- `value`: The new value.

readOnly()

If the edit box is read-only or editable by the user.

setReadOnly(*value*)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

`text`()
The current text.

`setText`(*value*)
Setter for *text()*.

Parameters

- `value`: The new value.

`hintText`()
The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText`(*value*)
Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems`()
A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(*value*)
Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()
How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(*value*)
Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()
If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(*value*)
Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)
Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

textSelection()
Returns the current text selection indices.

OnChanged
Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

OnPopupTextSelected
Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)
General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)
Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()

Sets the focus to this widget.

`isAlive`()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth`()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth`()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)
Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(*value*)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(*value*)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(*value*)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(*value*)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(*value*)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(*value*)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(*value*)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(*value*)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(*value*)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(*value*)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(*value*)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(*value*)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(*value*)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(*value*)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.49 Class clove::PasswordEditBox

`class clove::PasswordEditBox` : **public** *clove*::*EditBox*

A box for password input from the user.

Public Functions

`readOnly``()`

If the edit box is read-only or editable by the user.

`setReadOnly``(value)`

Setter for *readOnly()*.

Parameters

- `value`: The new value.

`text``()`

The current text.

`setText``(value)`

Setter for *text()*.

Parameters

- `value`: The new value.

`hintText``()`

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

`setHintText``(value)`

Setter for *hintText()*.

Parameters

- `value`: The new value.

`autocompletionItems`()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

`setAutocompletionItems`(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()

List of the children *clove::Widget* instances.

parentWidget ()

The parent *clove::Widget*.

name ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled ()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled (value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass ()

Custom css class(es).

setStyleClass (value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style ()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

`strictHorizontalSizing``()`

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

`setStrictHorizontalSizing``(value)`

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing``()`

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing``(value)`

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch``()`

Alias for *verticalStretchAffinity()*.

`setVstretch``(value)`

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch``()`

Alias for *horizontalStretchAffinity()*.

`setHstretch``(value)`

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy``()`

If the widget is in busy state, typically resulting in a loading animation.

`setBusy``(value)`

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.50 Class clove::PopupMenu

class *clove*::**PopupMenu** : **public** *clove*::*AbstractMenu*

A popup menu (i.e. a box with vertically listed actions).

Public Functions

expanderIndicatorDirection()

If the expander indicator is to be shown on the `'right'` or `'left'` side.

setExpanderIndicatorDirection(value)

Setter for *expanderIndicatorDirection()*.

Parameters

- `value`: The new value.

static show(menu, showconfig)

Shows a popup menu.

The root view of `menu` is expected to be a *clove::PopupMenu* (or a subclass).

Parameters

- `menu`: The widget configuration, as for *clove::build()*, which specifies the popup menu.
- `showconfig`: A configuration object which specifies some presentation aspects.

actions()

The actions to be shown in this menu.

It is a list of action configuration objects, each having a structure like `{name:'myaction', label:'My menu action'}`. It can have a list of sub-items in the` subactions` property.

Instead of a simple list, it may also be a *clove::Datasource* with the action configuration structures aligned in rows.

Use *clove::MenuSeparator* for a separator.

One action allows this properties:

- `name`: The action name.
- `label`: The label string.
- `icon`: A *clove::Icon*.
- `disabled`: If it is disabled.
- `invisible`: If it is invisible.
- `checkable`: If it is checkable.
- `checked`: If it is checked.

setActions(value)

Setter for *actions()*.

Parameters

- `value`: The new value.

OnActionTriggered

Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

OnBeforeSubactionsExpanded

Triggered just before a branch of subaction is expanded.

Implement this event for populating it dynamically.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit` `()`

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly` `()`

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize` `()`

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize` `()`

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout` `()`

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus` `()`

Sets the focus to this widget.

`isAlive` `()`

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth` `()`

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth` `()`

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computeMinimalHeightForWidth` `(w)`

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.51 Class clove::PopupMenuButton

`class clove::PopupMenuButton` : **`public`** *clove::Button*

A special *clove::Button* which opens a popup menu when it is triggered.

Public Functions

`actions`()

The menu actions. Same as for *clove::Menubar::actions()*.

`setActions`(value)

Setter for *actions()*.

Parameters

- `value`: The new value.

`OnActionTriggered`

Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

`label`()

The label text.

`setLabel`(value)

Setter for *label()*.

Parameters

- `value`: The new value.

`icon`()

The optional *clove::Icon* button icon.

`setIcon`(value)

Setter for *icon()*.

Parameters

- `value`: The new value.

checkable()

If the button is checkable (i.e. has a checked-flag).

If it has, the checked-flag is controlled by *checked()*.

setCheckable(value)

Setter for *checkable()*.

Parameters

- `value`: The new value.

checked()

For a checkable button, return if it is checked.

See also *checkable()*.

User interactions do not toggle the checked flag by default. This must be scripted in own event handlers as needed.

setChecked(value)

Setter for *checked()*.

Parameters

- `value`: The new value.

OnClicked

Triggered when the user clicks on this button.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()

Sets the focus to this widget.

isAlive()
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)
Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)
Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.52 Class clove::ProgressBar

`class` *`clove`*`::`**`ProgressBar`** : **`public`** *clove::Widget*
A progress bar.

Public Functions

`value``()`
The progress value between `0.0` and `1.0`; or `undefined` for indeterminate progress.

`setValue``(`*value*`)`
Setter for *value()*.

Parameters

- `value`: The new value.

`orientation``()`
If to present the progress bar `'vertical'`ly or `'horizontal'`ly.

`setOrientation``(`*value*`)`
Setter for *orientation()*.

Parameters

- `value`: The new value.

label()

An optional additional label text, which is displayed combined with the progress value.

setLabel(*value*)

Setter for *label()*.

Parameters

- `value`: The new value.

labelFunction()

A `function(value, widget)` which returns a custom label text.

setLabelFunction(*value*)

Setter for *labelFunction()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be ‘undefined’ for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()

Sets the focus to this widget.

isAlive ()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth ()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass (clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()

The namescope this widget contains to.

nameScope ()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()

List of the children *clove::Widget* instances.

parentWidget ()

The parent *clove::Widget*.

name ()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (*value*)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled()`

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled``(`*value*`)`

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass()`

Custom css class(es).

`setStyleClass``(`*value*`)`

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style()`

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle``(`*value*`)`

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility()`

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility``(`*value*`)`

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing()`

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing``(`*value*`)`

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(*value*)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(*value*)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(*value*)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(*value*)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(*value*)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(*value*)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(*value*)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(*value*)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.53 Class clove::ProxyDatasource

`class` `clove`::**`ProxyDatasource`** : **`public`** *clove*::*Datasource*, **`public`** *clove*::*Headersource*

A *clove::Datasource* implementation which proxies the content of another datasource in a somehow transformed way.

Note: This class is an abstract base class. Use one of the subclasses.

Subclassed by *clove::FilterProxyDatasource*, *clove::SortProxyDatasource*

Public Functions

`setDatasource`(datasource)

Sets the source datasource.

Parameters

- `datasource`: The source *clove::Datasource*.

`proxyPointerToNativePointer`(proxyptr)

Translates a *clove::DatasourceValuePointer* from this proxy datasource to a one for the source datasource.

Parameters

- `proxyptr`: A value pointer.

`nativePointerToProxyPointer`(nativeptr)

Translates a *clove::DatasourceValuePointer* from the source datasource to a one for this proxy datasource.

Parameters

- `nativeptr`: A value pointer.

refresh()

Refreshes the filtering.

Call this when the outer conditions changed and the filters must be applied again.

getValue(ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata(ptr)

Returns an object of metadata information (like icons, status infos used for styling, . . .). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

changeValue(ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount(*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount(*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer(irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

`parent` (ptr)
Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

`valuePointerNavigateInDepth` (ptr, direction, mayexpandfct)
Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

`OnDataInsert`
Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataRemove`
Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

`OnDataUpdate`
Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

`getRowHeader` (irow, *parent*)
Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`getColumnHeader` (irow, *parent*)
Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

`rowHeadersVisible` (*parent*)
If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`columnHeadersVisible` (*parent*)
If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

`OnHeaderDataInsert`
Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataRemove`
Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderDataUpdate`
Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

`OnHeaderVisibilityUpdated`
Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

18.1.54 Class clove::RadioButton

`class` `clove`**`::RadioButton`** : **`public`** *clove::Widget*
A radio button.

Typically some radio buttons are assigned to one *clove::RadioGroup* so the user can choose one of them (while a *clove::CheckButton* allows 0..n choices).

Public Functions

`label``()`
The label text.

`setLabel``(value)`
Setter for *label()*.

Parameters

- `value`: The new value.

`checked``()`
If this radio button is checked.

See also *clove::RadioGroup::selected()*.

`setChecked``(value)`
Setter for *checked()*.

Parameters

- `value`: The new value.

`group``()`
The *clove::RadioGroup* this radio button belongs to.

Note: It is possible to assign strings to it, see *clove::RadioGroup::getGroupByPublicName()*.

setGroup(value)

Setter for *group()*.

Parameters

- `value`: The new value.

groupValue()

The currently selected value of the radiogroup this button belongs to. See *group()*.

setGroupValue(value)

Setter for *groupValue()*.

Parameters

- `value`: The new value.

valueDef()

The value this button is representing. You can use this together with *clove::RadioGroup::selectedValue()*.

setValueDef(value)

Setter for *valueDef()*.

Parameters

- `value`: The new value.

OnChanged

Triggered when this radio button was selected.

This is a *clove.Event* instance. See Manual for details.

OnClicked

Triggered when this radio button was selected. Same as OnChanged.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.55 Class clove::RadioGroup

`class clove::RadioGroup`

Groups some *clove::RadioButton* together in a logical way, so the user can select exactly one of them.

Assign buttons to a group by setting *clove::RadioButton::group()*.

Public Functions

`RadioGroup``()`

`selected``()`

Returns the *clove::RadioButton* which is currently selected in this group.

`select``(w)`

Selects a button in the group.

Parameters

- `w`: The *clove::RadioButton* to select.

`OnChanged`

Triggered when another button was selected in this group.

This is a *clove.Event* instance. See Manual for details.

`static getGroupByPublicName``(name)`

Returns a *clove::RadioGroup* by name. Either returns an already existing one, or creates a new one.

Developers of library functionality should not use this function due to risks of naming clashes.

Parameters

- `name`: The group name.

`selectedIndex``()`

Returns the position index of the currently selected child in this group.

`selectedValue``()`

Returns the value of the currently selected child in this group.

`selectByIndex``(i)`

Selects a child by its position index.

Parameters

- `i`: The position index.

`selectByValue``(v)`

Selects a child by its value. It will also understand position indexes as fallback.

Parameters

- `v`: The value.

`unselectAll``()`

Unselects all childs in this group.

18.1.56 Class clove::RawResizeSplitter

`class` `clove::`**`RawResizeSplitter`** : **`public`** *clove*::*Widget*

The actual splitter in a *clove::ResizeSplitter*.

Not intended for direct usage in a user interface!

Public Functions

`OnMoveBegin`

Triggered when the user begins to move the splitter.

This is a *clove.Event* instance. See Manual for details.

`OnMove`

Triggered subsequently while the user moves the splitter.

This is a *clove.Event* instance. See Manual for details.

`OnMoveEnd`

Triggered when the user ends moving the splitter.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.57 Class clove::ResizeSplitter

`class` *`clove`*`::`**`ResizeSplitter`** : **`public`** *clove*::*Widget*

A container for multiple widgets with splitters for manual resizing between them.

Public Functions

`orientation`()

If the two widgets are stacked `'vertical'`ly or `'horizontal'`ly.

`setOrientation`(value)

Setter for *orientation()*.

Parameters

- `value`: The new value.

`body`()

The list of body widgets (i.e. the widgets to show separated by splitters) as widget configurations like for *clove::build()*.

`setBody`(value)

Setter for *body()*.

Parameters

- `value`: The new value.

`bodyWidgets`()

Returns the list of all body widgets as *clove::Widget* instances.

`setBodyWidget`(i, config)

Sets (overrides) the body widget at a given position.

Parameters

- `i`: The widget position.
- `config`: The widget as widget configurations like for *clove::build()*.

`addBodyWidget`(config)

Adds a new body widget to the end.

Parameters

- `config`: The widget as widget configurations like for *clove::build()*.

`insertBodyWidget`(i, config)

Inserts a new body widget somewhere.

Parameters

- `i`: The widget position.
- `config`: The widget as widget configurations like for *clove::build()*.

sizeFractions()

The sizes of each widget inside this resize splitter as list of numbers. It has a positive entry for each widget, describing the amount of room it takes. The values have just relative meaning (and are typically fractions of 1).

setSizeFractions(value)

Setter for *sizeFractions()*.

Parameters

- `value`: The new value.

showOnly()

Specifies if to show just one of the two sides (`0`, `1`, `undefined`).

setShowOnly(value)

Setter for *showOnly()*.

Parameters

- `value`: The new value.

splitterWidth()

The splitter width (height for vertical orientation) as css length.

setSplitterWidth(value)

Setter for *splitterWidth()*.

Parameters

- `value`: The new value.

splitterVisibility()

The visibility of the splitter. See *clove::Widget::visibility()* for details.

setSplitterVisibility(value)

Setter for *splitterVisibility()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *`horizontalStretchAffinity()`*`==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.58 Class clove::RichEdit

class *clove*::**RichEdit** : **public** *clove*::*Widget*

A user input box for text with rich formatting.

This widget includes formatting toolbars. For a bare scriptable edit box, see *clove::RichEditBox*.

Public Functions

htmlContent()

The content text as html string.

setHtmlContent(value)

Setter for *htmlContent()*.

Parameters

- `value`: The new value.

richeditbox()

Returns the inner *clove::RichEditBox*.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be ‘undefined’ for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout ()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus ()

Sets the focus to this widget.

isAlive ()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth ()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.59 Class clove::RichEditBox

`class` `clove`**`::RichEditBox`** : **`public`** *clove::Widget*

A user input box for text with rich formatting.

This widget allows to be controlled from external toolbars. For a widget including a formatting toolbar, see *clove::RichEdit*.

Public Functions

`htmlContent``()`

The content text as html string.

`setHtmlContent``(value)`

Setter for *htmlContent()*.

Parameters

- `value`: The new value.

`selection``()`

The current selection (as a web-api Range object).

`setSelection``(value)`

Setter for *selection()*.

Parameters

- `value`: The new value.

contentHtmlNode()
Returns the html content dom node.

toggleSelectionBold()
Toggles the bold-flag for the current selection.

toggleSelectionItalic()
Toggles the italic-flag for the current selection.

toggleSelectionUnderline()
Toggles the underline-flag for the current selection.

selectionIncreaseFontSize()
Increases the font size for the current selection.

selectionDecreaseFontSize()
Decreases the font size for the current selection.

selectionSetForegroundColor(c)
Sets the foreground color for the current selection.

Parameters

- `c`: The color string.

selectionSetBackgroundColor(c)
Sets the background color for the current selection.

Parameters

- `c`: The color string.

selectionInsertList(config)
Inserts a list at the current place.

config may contain:

- `ordered`: If the list is ordered.

  Parameters

  - `config`: The list configuration.

declareProperty(k, defaultV)
Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)
General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly ()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize ()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize ()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout`()`
Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus`()`
Sets the focus to this widget.

isAlive`()`
Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth`()`
Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth`()`
Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth`(w)`
Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth`(w)`
Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth`()`
Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth`()`
Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth`(w)`
Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility``()`
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets``()`
List of the children *clove::Widget* instances.

`parentWidget``()`
The parent *clove::Widget*.

`name``()`
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)
Setter for *name()*.

Parameters

- `value`: The new value.

`enabled``()`
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass``()`
Custom css class(es).

`setStyleClass`(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style``()`
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)
Setter for *style()*.

Parameters

- `value`: The new value.

`visibility``()`
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *`verticalStretchAffinity()`*`==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.60 Class clove::RootNameScope

class *clove*::**RootNameScope** : **public** *clove*::*NameScope*
A standalone namescope.

Public Functions

`getByName`(name)
Finds a widget by name within this namescope.

Parameters

- `name`: The widget name.

`addChildNameScope`(childnamescope)
Adds a namescope to the children of this one, so a searcher will also search in this child.

Parameters

- `childnamescope`: The child namescope.

18.1.61 Class clove::ScrollView

`class` *`clove`*`::`**`ScrollView`** : **`public`** *clove::Widget*
A container for making the children scrollable.

Public Functions

`body`()
The inner widget as widget configuration like for *clove::build()*.

`setBody`(value)
Setter for *body()*.

Parameters

- `value`: The new value.

`bodyWidget`()
Returns the body as *clove::Widget*.

`bodySize`()
The (outer) size of the body widget.

`verticalScrollPosition`()
The vertical scroll position.

`setVerticalScrollPosition`(value)
Setter for *verticalScrollPosition()*.

Parameters

- `value`: The new value.

`horizontalScrollPosition`()
The horizontal scroll position.

`setHorizontalScrollPosition`(value)
Setter for *horizontalScrollPosition()*.

Parameters

- `value`: The new value.

declareProperty (k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty (k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty (k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty (k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init (rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit ()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()
The namescope this widget contains to.

nameScope()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()
List of the children *clove::Widget* instances.

parentWidget()
The parent *clove::Widget*.

name()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()
Custom css class(es).

setStyleClass(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()
Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()
If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()
Returns inner width and height of this widget.

`outerSize`()
Returns outer width and height of this widget.

`OnDestroyed`
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.62 Class clove::Slider

`class` *`clove`*`::`**`Slider`** : **`public`** *clove*::*Widget*

A slider allows the user to choose a number from a given value interval.

The user chooses a value by moving a handle alongside a line.

Public Functions

`min``()`

The minimum value.

`setMin``(`*value*`)`

Setter for *min()*.

Parameters

- `value`: The new value.

`max``()`

The maximum value.

`setMax``(`*value*`)`

Setter for *max()*.

Parameters

- `value`: The new value.

`stepsize``()`

The step size.

`setStepsize``(`*value*`)`

Setter for *stepsize()*.

Parameters

- `value`: The new value.

`value``()`

The current slider value.

`setValue``(`*value*`)`

Setter for *value()*.

Parameters

- `value`: The new value.

`orientation``()`

If to present the slider `'vertical'`ly or `'horizontal'`ly.

`setOrientation`(*value*)

Setter for *orientation()*.

Parameters

- `value`: The new value.

`OnChanged`

Triggered when the slider value changed.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(*value*)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(*value*)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()
Custom css class(es).

`setStyleClass`(*value*)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(*value*)
Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(*value*)
Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(*value*)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(*value*)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(*value*)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(*value*)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(*value*)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(*value*)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(*value*)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(*value*)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(*value*)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.63 Class clove::SortProxyDatasource

`class` `clove`::**`SortProxyDatasource`** : **`public`** *clove::ProxyDatasource*

A *clove::Datasource* implementation which sorts another *clove::Datasource*.

Public Functions

`SortProxyDatasource`(datasource, rowcomparefct, colcomparefct)

Parameters

- `datasource`: The source *clove::Datasource*.
- `rowcomparefct`: A `function(datasource, irow, jrow, parent)` returning a negative value if `irow` points to a row lower than `jrow`, a positive value if it is greater, or 0 if both are equal.
- `colcomparefct`: A `function(datasource, icol, jcol, parent)` returning a negative value if `icol` points to a column lower than `jcol`, a positive value if it is greater, or 0 if both are equal.

`setRowComparator`(rowcomparefct)

Sets the row comparator function.

Parameters

- `rowcomparefct`: The row comparator function. See constructor for details.

`setColumnComparator`(colcomparefct)

Sets the column comparator function.

Parameters

- `colcomparefct`: The column comparator function. See constructor for details.

`setDatasource`(datasource)

Sets the source datasource.

Parameters

- `datasource`: The source *clove::Datasource*.

proxyPointerToNativePointer(proxyptr)

Translates a *clove::DatasourceValuePointer* from this proxy datasource to a one for the source datasource.

Parameters

- `proxyptr`: A value pointer.

nativePointerToProxyPointer(nativeptr)

Translates a *clove::DatasourceValuePointer* from the source datasource to a one for this proxy datasource.

Parameters

- `nativeptr`: A value pointer.

refresh()

Refreshes the filtering.

Call this when the outer conditions changed and the filters must be applied again.

getValue(ptr)

Returns the value for a given node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

getMetadata(ptr)

Returns an object of metadata information (like icons, status infos used for styling, …). Optional.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

changeValue(ptr, value)

Change the value for a given node.

This method is called from external places, e.g. when a user makes changes in a datasource-connected widget.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `value`: The new value.

rowCount(*parent*)

Returns the number of rows for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

columnCount(*parent*)

Returns the number of columns for a given node.

Parameters

- `parent`: The parent as *clove::DatasourceValuePointer*.

valuePointer (irow, icol, *parent*)

Constructs and returns a *clove::DatasourceValuePointer* for a given node.

Parameters

- `irow`: The row index.
- `icol`: The column index.
- `parent`: The parent as *clove::DatasourceValuePointer*.

parent (ptr)

Returns the parent node for a given node.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.

valuePointerNavigateInDepth (ptr, direction, mayexpandfct)

Returns a *clove::DatasourceValuePointer* resulting in the original one by navigation in depth.

Parameters

- `ptr`: A node as *clove::DatasourceValuePointer*.
- `direction`: The direction (+1 or -1).
- `mayexpandfct`: A `function(ptr)` which returns `true` iff this node's children are to be traversed.

OnDataInsert

Triggered when a node insertion takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataRemove

Triggered when a node removal takes place.

This is a *clove.Event* instance. See Manual for details.

OnDataUpdate

Triggered when a node data update takes place.

This is a *clove.Event* instance. See Manual for details.

getRowHeader (irow, *parent*)

Returns the header configuration for a given row.

Parameters

- `irow`: The row index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

getColumnHeader (irow, *parent*)

Returns the header configuration for a given column.

Parameters

- `irow`: The column index.
- `parent`: The parent node as *clove::DatasourceValuePointer*.

rowHeadersVisible(*parent*)

If row headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

columnHeadersVisible(*parent*)

If column headers are visible.

Parameters

- `parent`: The parent node as *clove::DatasourceValuePointer*.

OnHeaderDataInsert

Triggered when a new row or column of header data was inserted.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataRemove

Triggered when a row or column of header data was removed.

This is a *clove.Event* instance. See Manual for details.

OnHeaderDataUpdate

Triggered when header data were updated.

This is a *clove.Event* instance. See Manual for details.

OnHeaderVisibilityUpdated

Triggered when header visibilities changed.

This is a *clove.Event* instance. See Manual for details.

18.1.64 Class clove::Spacer

class *clove*::**Spacer** : **public** *clove*::*Widget*

A spacer widget for filling gaps in a layout.

It is essentially just a widget which does nothing, but respects the layouting properties like all widgets. Add them to a layout and configure them according to your sizing requirements.

Public Functions

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus``()`

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize``()`

Returns inner width and height of this widget.

`outerSize``()`

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.65 Class clove::StackLayout

`class clove::StackLayout` : **`public`** *clove::GridLayout*

A mixin for stack layout implementations.

Subclassed by *clove::HorizontalStack*, *clove::VerticalStack*

Public Functions

`insertRow`(i)

Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

`insertColumn`(i)

Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

`removeRow`(i)

Removes a row from the grid.

Parameters

- `i`: The row index.

`removeColumn`(i)

Removes a column from the grid.

Parameters

- `i`: The column index.

`children`()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)

Setter for *children()*.

Parameters

- `value`: The new value.

`addChild`(value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

`clearChilds`()

Removes all childs from the layout.

18.1.66 Class clove::TabView

class *clove*::**TabView** : **public** *clove*::*Widget*

A tabbed container.

Typically used for grouping and folding some user interface parts together for better space efficiency and higher usability.

Subclassed by *clove::Carousel*

Public Functions

tabs()

The list of tabs.

Each tab is a widget configuration, like for *clove::build()*, with some additional optional keys which holds infos like the tab header text. So, for example, one item in that list could be `{view:'Something', ..., tabLabel:'Tab1'}`. See also *addTab()*.

setTabs(value)

Setter for *tabs()*.

Parameters

- `value`: The new value.

currentTab()

The index of the currently selected tab.

setCurrentTab(value)

Setter for *currentTab()*.

Parameters

- `value`: The new value.

userMayAddTabs()

If to show a button for adding new tabs.

If `true`, implement a handler for OnTabCreationRequested and create a tab with *addTab()* there.

setUserMayAddTabs(value)

Setter for *userMayAddTabs()*.

Parameters

- `value`: The new value.

tabBarLocation()

Where to place the tab bar.

Either `'top'`, `'left'`, `'bottom'` or `'right'`.

setTabBarLocation(value)

Setter for *tabBarLocation()*.

Parameters

- `value`: The new value.

switchInvisibleAnimationName()

Optional animation name for switching away from a tab.

setSwitchInvisibleAnimationName(value)

Setter for *switchInvisibleAnimationName()*.

Parameters

- `value`: The new value.

switchVisibleAnimationName()

Optional animation name for switching to a tab.

setSwitchVisibleAnimationName(value)

Setter for *switchVisibleAnimationName()*.

Parameters

- `value`: The new value.

switchInvisibleAnimationDuration()

Optional animation duration (in msec) for the switch away animation.

See also *clove::TabView::switchInvisibleAnimationName()*.

setSwitchInvisibleAnimationDuration(value)

Setter for *switchInvisibleAnimationDuration()*.

Parameters

- `value`: The new value.

switchVisibleAnimationDuration()

Optional animation duration (in msec) for the switch animation.

See also *clove::TabView::switchVisibleAnimationName()*.

setSwitchVisibleAnimationDuration(value)

Setter for *switchVisibleAnimationDuration()*.

Parameters

- `value`: The new value.

OnTabCreationRequested

Triggered when the user requested a new tab.

See also *userMayAddTabs()*.

This is a *clove.Event* instance. See Manual for details.

addTab(config)

Adds a new tab.

The configuration may contain the following additional keys:

- `tabLabel`: The tab header text.
- `tabIcon`: The tab icon as *clove::Icon*.
- `tabMayBeClosedByUser`: If the user may close this tab.

This method returns a *clove::Widget* which can be used for getting subwidgets or removing the tab.

Parameters

- `config`: A widget configuration like for *clove::build()*.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()
Alias for *horizontalStretchAffinity()*.

setHstretch(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.67 Class clove::TableView

`class` `clove`**`::TableView`** : **`public`** *clove::DataView*

A view which shows a *clove::Datasource* in a table.

Public Functions

`datasource``()`

The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

`setDatasource``(value)`

Setter for *datasource()*.

Parameters

- `value`: The new value.

`dataViewProcessor``()`

An optional `function(ptr, value)` for processing a datasource value to a display string.

`setDataViewProcessor``(value)`

Setter for *dataViewProcessor()*.

Parameters

- `value`: The new value.

`cellGenerator``()`

A generator function for complex cell content.

This method is called for each cell with `(clove::Widget, clove::DatasourceValuePointer)`. It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

`setCellGenerator``(value)`

Setter for *cellGenerator()*.

Parameters

- `value`: The new value.

`alwaysAllocateExpanderSpace``()`

If some space is allocated for an expander even for cells without children.

`setAlwaysAllocateExpanderSpace``(value)`

Setter for *alwaysAllocateExpanderSpace()*.

Parameters

- `value`: The new value.

showOnlyFirstColumn()
If to show only the first column and hide the other ones.

setShowOnlyFirstColumn(value)
Setter for *showOnlyFirstColumn()*.

Parameters

- `value`: The new value.

hideExpanders()
If to hide all expanders.

setHideExpanders(value)
Setter for *hideExpanders()*.

Parameters

- `value`: The new value.

allowSelection()
If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

setAllowSelection(value)
Setter for *allowSelection()*.

Parameters

- `value`: The new value.

allowChecking()
If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

setAllowChecking(value)
Setter for *allowChecking()*.

Parameters

- `value`: The new value.

granularity()
The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

setGranularity(value)
Setter for *granularity()*.

Parameters

- `value`: The new value.

showChangeMenu()

If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:

- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

setShowChangeMenu(value)

Setter for *showChangeMenu()*.

Parameters

- `value`: The new value.

editOnGesture()

If the user shall be able to edit cells by double clicking/tapping them.

setEditOnGesture(value)

Setter for *editOnGesture()*.

Parameters

- `value`: The new value.

rowsResizable()

If the user shall be able to resize rows.

setRowsResizable(value)

Setter for *rowsResizable()*.

Parameters

- `value`: The new value.

columnsResizable()

If the user shall be able to resize columns.

setColumnsResizable(value)

Setter for *columnsResizable()*.

Parameters

- `value`: The new value.

selectCell(ptr)

Selects a cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

isCellSelected(ptr)

Checks if a given cell is selected.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`checkedCells`()
Returns the checked cells as list of *clove::DatasourceValuePointer*.

`setCheckedCells`(ptrs)
Seturns the checked cells.

Parameters

- `ptrs`: A list of *clove::DatasourceValuePointer*.

`isCellChecked`(ptr)
Checks if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`setCellChecked`(ptr, val)
Sets if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

`selection`()
Returns the selected item as *clove::DatasourceValuePointer*.

`headersource`()
The *clove::Headersource* providing row and column header configurations.

`setHeadersource`(value)
Setter for *headersource()*.

Parameters

- `value`: The new value.

`gridVisible`()
If to show a cell grid.

`setGridVisible`(value)
Setter for *gridVisible()*.

Parameters

- `value`: The new value.

`nodeActivationNeedsDoubleClick`()
If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

`setNodeActivationNeedsDoubleClick`(value)
Setter for *nodeActivationNeedsDoubleClick()*.

Parameters

- `value`: The new value.

`expandCell`(ptr)

Expands a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`expandCellRecursive`(ptr)

Expands a given cell and all parents.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`collapseCell`(ptr)

Collapses a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`isCellExpanded`(ptr)

Checks if a given cell is expanded.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`editCell`(ptr)

Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`OnSelectionChanged`

Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`getPreferredHeightForWidth`(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

`addStyleClass`(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

`removeStyleClass`(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

`isStyleClass`(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

`setStyleClassAssigned`(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

`containingNameScope`()

The namescope this widget contains to.

`nameScope`()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

`remove`(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.68 Class clove::TimeBox

class *clove*::**TimeBox** : **public** *clove*::*EditBox*

A user input box for time input.

The actual behavior depends on the browser.

Public Functions

time()

The time value in the box as JavaScript Date.

setTime(value)

Setter for *time()*.

Parameters

- `value`: The new value.

readOnly()

If the edit box is read-only or editable by the user.

setReadOnly(value)

Setter for *readOnly()*.

Parameters

- `value`: The new value.

text()

The current text.

setText(value)

Setter for *text()*.

Parameters

- `value`: The new value.

hintText()

The hint text.

Typically contains something like a label text. It is shown as a hint when the box is empty.

setHintText(value)

Setter for *hintText()*.

Parameters

- `value`: The new value.

autocompletionItems()

A *clove.Datasource* with a list of autocompletion items to popup.

It is allowed to observe the `text` in order to generate live content for this list.

setAutocompletionItems(value)

Setter for *autocompletionItems()*.

Parameters

- `value`: The new value.

`autocompletionFilter`()

How to filter the autocompletion list.

Either `'startswith'`, `'contains'`, `function(itemtext, boxtext)` or `undefined`.

`setAutocompletionFilter`(value)

Setter for *autocompletionFilter()*.

Parameters

- `value`: The new value.

`autocompletionOpenForNoText`()

If to show the autocompletion list when the edit box is empty.

`setAutocompletionOpenForNoText`(value)

Setter for *autocompletionOpenForNoText()*.

Parameters

- `value`: The new value.

`setTextSelection`(ifrom, ito)

Selects the specified part of the text.

Parameters

- `ifrom`: The selection begin index.
- `ito`: The selection end index.

`textSelection`()

Returns the current text selection indices.

`OnChanged`

Triggered when the text was changed.

This is a *clove.Event* instance. See Manual for details.

`OnPopupTextSelected`

Triggered when a text was selected from the popup.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()
If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()
List of the children *clove::Widget* instances.

`parentWidget`()
The parent *clove::Widget*.

`name`()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)
Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()
If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)
Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()
Custom css class(es).

`setStyleClass`(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)
Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with *horizontalStretchAffinity()*==0.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with *verticalStretchAffinity()*==0.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.69 Class clove::Toolbar

class *clove*::**Toolbar** : **public** *clove::Widget*

A toolbar has header texts and a menu bar in a perceptible visual bar, typically used on top of a user interface with all available width.

Public Functions

actions()

Menu actions.

See *clove::Menubar::actions()*.

setActions(value)

Setter for *actions()*.

Parameters

- `value`: The new value.

head1()

The 1st header text.

setHead1(value)

Setter for *head1()*.

Parameters

- `value`: The new value.

head2()

The 2nd header text.

setHead2(value)

Setter for *head2()*.

Parameters

- `value`: The new value.

headControl()

An optional widget for arbitrary control purposes as widget configuration like for *clove::build()*. It's displayed below the menu bar in full width.

setHeadControl(value)

Setter for *headControl()*.

Parameters

- `value`: The new value.

headControlWidget()

Returns the control widget as *clove::Widget* (or undefined if no *headControl()* is set).

icon()

The header icon as *clove::Icon*.

setIcon(value)

Setter for *icon()*.

Parameters

- `value`: The new value.

`OnActionTriggered`

Triggered when a menu action was chosen by the user for execution.

The event arguments contain the selected action name in `action`.

This is a *clove.Event* instance. See Manual for details.

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namespace to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()
Custom css class(es).

`setStyleClass`(value)
Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()
Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)
Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()
If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)
Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()
If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()
If the widget can have the keyboard focus.

`setMayFocus`(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()
If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.70 Class clove::TreeView

`class clove::TreeView` : **`public`** *clove::DataView*

A view which shows a *clove::Datasource* in a tree.

Public Functions

`datasource``()`

The *clove::Datasource* for this view.

This provides the actual data to display. See *clove::NativeDatasource* for a ready-to-use implementation.

`setDatasource``(value)`

Setter for *datasource()*.

Parameters

- `value`: The new value.

`dataViewProcessor``()`

An optional `function(ptr, value)` for processing a datasource value to a display string.

`setDataViewProcessor``(value)`

Setter for *dataViewProcessor()*.

Parameters

- `value`: The new value.

`cellGenerator``()`

A generator function for complex cell content.

This method is called for each cell with `(clove::Widget, clove::DatasourceValuePointer)`. It may return a widget configuration as for *clove::build()*. If it does, the cell is constructed with this widget as content. The content must define an `update(clove::Widget, clove::DatasourceValuePointer, value)` method, which takes the cell datasource value and configures the inner widget according to that.

`setCellGenerator``(value)`

Setter for *cellGenerator()*.

Parameters

- `value`: The new value.

alwaysAllocateExpanderSpace()
If some space is allocated for an expander even for cells without children.

setAlwaysAllocateExpanderSpace(value)
Setter for *alwaysAllocateExpanderSpace()*.

Parameters

- `value`: The new value.

showOnlyFirstColumn()
If to show only the first column and hide the other ones.

setShowOnlyFirstColumn(value)
Setter for *showOnlyFirstColumn()*.

Parameters

- `value`: The new value.

hideExpanders()
If to hide all expanders.

setHideExpanders(value)
Setter for *hideExpanders()*.

Parameters

- `value`: The new value.

allowSelection()
If it is allowed to select nodes.

This is always a single selection. For something like multi-selection, please check *allowChecking()*.

setAllowSelection(value)
Setter for *allowSelection()*.

Parameters

- `value`: The new value.

allowChecking()
If it is allowed to check cells.

This is a way to select 0..n data cells. For a single-selection, please check *allowSelection()*.

setAllowChecking(value)
Setter for *allowChecking()*.

Parameters

- `value`: The new value.

granularity()
The granularity for stuff like selection and checking.

Either `'cell'` or `'row'`.

`setGranularity`(value)
Setter for *granularity()*.

Parameters

- `value`: The new value.

`showChangeMenu`()
If a menu shall be shown for making manual changes to the data source.

Instead of true, it may also be a configuration object, which may contain the following:

- `item_foo_label`, `item_foo_icon`, `item_foo_isvisible`, `item_foo_action`: Specifies a menu action `foo` by four functions. Each function gets `widget, datasource` as parameters. Respectively they have to return a label, an icon, if it is actually visible, or have to execute the action. It is also possible to customize the existing actions `addrow`, `addrowbefore`, `addcolumn`, `addcolumnbefore`, `removerow`, `removecolumn` and `edit` this way.

`setShowChangeMenu`(value)
Setter for *showChangeMenu()*.

Parameters

- `value`: The new value.

`editOnGesture`()
If the user shall be able to edit cells by double clicking/tapping them.

`setEditOnGesture`(value)
Setter for *editOnGesture()*.

Parameters

- `value`: The new value.

`rowsResizable`()
If the user shall be able to resize rows.

`setRowsResizable`(value)
Setter for *rowsResizable()*.

Parameters

- `value`: The new value.

`columnsResizable`()
If the user shall be able to resize columns.

`setColumnsResizable`(value)
Setter for *columnsResizable()*.

Parameters

- `value`: The new value.

`selectCell`(ptr)
Selects a cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`isCellSelected`(ptr)
Checks if a given cell is selected.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`checkedCells`()
Returns the checked cells as list of *clove::DatasourceValuePointer*.

`setCheckedCells`(ptrs)
Seturns the checked cells.

Parameters

- `ptrs`: A list of *clove::DatasourceValuePointer*.

`isCellChecked`(ptr)
Checks if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

`setCellChecked`(ptr, val)
Sets if a given cell is checked.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.
- `val`: If the cells shall be checked.

`selection`()
Returns the selected item as *clove::DatasourceValuePointer*.

`headersource`()
The *clove::Headersource* providing row and column header configurations.

`setHeadersource`(value)
Setter for *headersource()*.

Parameters

- `value`: The new value.

`gridVisible`()
If to show a cell grid.

`setGridVisible`(value)
Setter for *gridVisible()*.

Parameters

- `value`: The new value.

nodeActivationNeedsDoubleClick()

If node activation needs a double-click.

Note: If you set this, you should find alternative ways for users of mobile devices!

setNodeActivationNeedsDoubleClick(value)

Setter for *nodeActivationNeedsDoubleClick()*.

Parameters

- `value`: The new value.

expandCell(ptr)

Expands a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

expandCellRecursive(ptr)

Expands a given cell and all parents.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

collapseCell(ptr)

Collapses a given cell.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

isCellExpanded(ptr)

Checks if a given cell is expanded.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

editCell(ptr)

Triggers the edit mode for a cell.

Only works if a method `_getdomcell(ir, ic, parent)` returns a dom node.

Parameters

- `ptr`: The *clove::DatasourceValuePointer*.

OnSelectionChanged

Triggered when the selection in the view changes.

This is a *clove.Event* instance. See Manual for details.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

init(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

doinit()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

doinitEarly()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

resize()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

`setDoStandaloneResizing`(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

`mayFocus`()

If the widget can have the keyboard focus.

`setMayFocus`(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

`horizontalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

`setHorizontalStretchAffinity`(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

`verticalStretchAffinity`()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

`setVerticalStretchAffinity`(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

`hasFocus`()

If the widget has the keyboard focus.

This also returns true if a child has focus.

`innerSize`()

Returns inner width and height of this widget.

`outerSize`()

Returns outer width and height of this widget.

`OnDestroyed`

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.71 Class clove::VerticalStack

class *clove*::**VerticalStack** : **public** *clove*::*Widget*, **public** *clove*::*StackLayout*

A container which stacks child widgets row-wise.

Public Functions

rows()

The list of child widgets as widget configurations like for *clove::build()*.

setRows(value)

Setter for *rows()*.

Parameters

- `value`: The new value.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for \`x.getProperty('name') `and`x.setName(v)\` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

bindProperty(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()

Sets the focus to this widget.

`isAlive`()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

`computeMinimalWidth`()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

`computePreferredWidth`()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass (clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()
The namescope this widget contains to.

nameScope ()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()
List of the children *clove::Widget* instances.

parentWidget ()
The parent *clove::Widget*.

name ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName (value)
Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()
Alias for *verticalStretchAffinity()*.

setVstretch(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch ()
Alias for *horizontalStretchAffinity()*.

setHstretch (value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy ()
If the widget is in busy state, typically resulting in a loading animation.

setBusy (value)
Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy ()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy (token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus ()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize ()
Returns inner width and height of this widget.

outerSize ()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

`insertRow`(i)

Inserts a new empty row to the grid.

Parameters

- `i`: The row index.

`insertColumn`(i)

Inserts a new empty column to the grid.

Parameters

- `i`: The column index.

`removeRow`(i)

Removes a row from the grid.

Parameters

- `i`: The row index.

`removeColumn`(i)

Removes a column from the grid.

Parameters

- `i`: The column index.

`children`()

List of all child widgets in this layout as widget configurations like in *clove::build()*.

`setChildren`(value)

Setter for *children()*.

Parameters

- `value`: The new value.

addChild (value)

Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds ()

Removes all childs from the layout.

18.1.72 Class clove::VideoPlayer

class *clove*::**VideoPlayer** : **public** *clove*::*MediaPlayer*

A widget which plays a video source and optionally allows user controls.

Public Functions

static canPlayType (t)

Determines if the browser can play a certain video type and returns a string as described in the html specs (->"canPlayType").

Parameters

- `t`: A mimetype string.

autoPlay ()

If to automatically start playback as soon as possible.

setAutoPlay (value)

Setter for *autoPlay()*.

Parameters

- `value`: The new value.

showControls ()

If to show user controls for play, pause and more.

setShowControls (value)

Setter for *showControls()*.

Parameters

- `value`: The new value.

currentMediaPosition ()

The current media playback position in seconds.

setCurrentMediaPosition (value)

Setter for *currentMediaPosition()*.

Parameters

- `value`: The new value.

loop()
If to playback in an endless loop.

setLoop(value)
Setter for *loop()*.

Parameters

- `value`: The new value.

muted()
If to mute the audio playback.

setMuted(value)
Setter for *muted()*.

Parameters

- `value`: The new value.

preload()
If to begin loading the media data as soon as possible.

setPreload(value)
Setter for *preload()*.

Parameters

- `value`: The new value.

volume()
The audio playback volume between `0.0` and `1.0`.

setVolume(value)
Setter for *volume()*.

Parameters

- `value`: The new value.

source()
The media source URL.

setSource(value)
Setter for *source()*.

Parameters

- `value`: The new value.

duration()
The duration of the loaded media source in seconds.

hasEnded()
If the playback has ended (i.e. reached the end).

isPaused()
If the playback is logically paused.

This does not return `true` just when loading stalls.

`nativeNode``()`

Returns the native html dom node.

`play``()`

Starts playback.

`pause``()`

Pauses playback.

`OnLoadingAborted`

Triggered when the media loading aborted for some reasons (e.g. network errors).

This is a *clove.Event* instance. See Manual for details.

`OnReadyForPlayback`

Triggered when there are enough data for starting playback.

This is a *clove.Event* instance. See Manual for details.

`OnDurationChanged`

Triggered when the playback duration changed.

See also *duration()*.

This is a *clove.Event* instance. See Manual for details.

`OnHasEnded`

Triggered when the playback has ended.

See also *hasEnded()*.

This is a *clove.Event* instance. See Manual for details.

`OnIsPaused`

Triggered when the playback logically paused.

This is not triggered when loading stalls.

This is a *clove.Event* instance. See Manual for details.

`OnIsPlaying`

Triggered when the playback logically starts.

This is not triggered when loading has stalled and resumes.

This is a *clove.Event* instance. See Manual for details.

`declareProperty``(k, defaultV)`

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty``(k)`

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

`effectivelyEnabled`()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

`effectiveVisibility`()

If this widget and all parent widgets are visible. See also *visibility()*.

`childrenWidgets`()

List of the children *clove::Widget* instances.

`parentWidget`()

The parent *clove::Widget*.

`name`()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)

Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()

If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)

Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()

Alias for *verticalStretchAffinity()*.

`setVstretch`(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()

Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()

If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.73 Class clove::Widget

class *clove*::**Widget**

Base class for a Clove widget.

Read the Manual about widgets.

Subclassed by *clove::AbstractMenu*, *clove::Border*, *clove::Button*, *clove::CheckButton*, *clove::DataView*, *clove::Dialog*, *clove::EditBox*, *clove::Expander*, *clove::FlatView*, *clove::Form*, *clove::Grid*, *clove::HorizontalStack*, *clove::HtmlView*, *clove::IconView*, *clove::ImageView*, *clove::Label*, *clove::MainView*, *clove::MediaPlayer*, *clove::ModalPanel*, *clove::ProgressBar*, *clove::RadioButton*, *clove::RawResizeSplitter*, *clove::ResizeSplitter*, *clove::RichEdit*, *clove::RichEditBox*, *clove::ScrollView*, *clove::Slider*, *clove::Spacer*, *clove::TabView*, *clove::Toolbar*, *clove::VerticalStack*, *clove::Wrap*

Public Functions

Widget(config, domnode)

Note: All widgets have a constructor with this signature. The configuration parameters are stored in `config`.

It is typically not required to call those constructors directly. Use *clove::build()* or some other factory functions (for special situations) instead.

Parameters

- `config`: A widget configuration as in *clove::build()*.
- `domnode`: The dom node to use as host for this widget.

declareProperty(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

getProperty(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

setProperty(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

`doresize`()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

`relayout`()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

`focus`()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass (clss)
Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass (clss)
Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned (clss, assigned)
Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope ()
The namescope this widget contains to.

nameScope ()
The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove (removeconfig)
Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled ()
If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility ()
If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets ()
List of the children *clove::Widget* instances.

parentWidget ()
The parent *clove::Widget*.

name ()
The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

`setName`(value)

Setter for *name()*.

Parameters

- `value`: The new value.

`enabled`()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

`setEnabled`(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

`styleClass`()

Custom css class(es).

`setStyleClass`(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

`style`()

Custom css style string. You should not use that, but *styleClass()* instead.

`setStyle`(value)

Setter for *style()*.

Parameters

- `value`: The new value.

`visibility`()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

`setVisibility`(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

`doStandaloneResizing`()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)
Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()
If the widget can have the keyboard focus.

setMayFocus(value)
Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)
Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()
Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)
Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()
If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

setStrictHorizontalSizing(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

strictVerticalSizing()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

setStrictVerticalSizing(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

vstretch()

Alias for *verticalStretchAffinity()*.

setVstretch(value)

Setter for *vstretch()*.

Parameters

- `value`: The new value.

hstretch()

Alias for *horizontalStretchAffinity()*.

setHstretch(value)

Setter for *hstretch()*.

Parameters

- `value`: The new value.

busy()

If the widget is in busy state, typically resulting in a loading animation.

setBusy(value)

Setter for *busy()*.

Parameters

- `value`: The new value.

registerBusy()

Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

unregisterBusy(token)

Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()

If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()

Returns inner width and height of this widget.

outerSize()

Returns outer width and height of this widget.

OnDestroyed

Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

`OnResized`

Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

`OnVisibilityChanged`

Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

`OnPropertyChanged`

Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

`OnKeyDown`

Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

`OnKeyUp`

Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

`OnKeyPress`

Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

18.1.74 Class clove::Wrap

`class` `clove::`**`Wrap`** : **`public`** *clove::Widget*, **`public`** *clove::WrapLayout*

A container for aligning child widgets in horizontal wrapping lines.

Public Functions

`declareProperty`(k, defaultV)

Declares a widget property.

This is intended to be called only from inside the widget constructor.

Read the Manual about widget properties and custom widgets.

Parameters

- `k`: The widget property name.
- `defaultV`: The default value.

`getProperty`(k)

General-purpose getter for widget properties.

Typically used for handling properties, which do not directly belong to the widget class but which are helpers for external aspects (e.g. a row index for positioning in a grid).

The known ones have a dedicated getter and setter, i.e. `x.name()` for `x.getProperty('name') `and`x.setName(v)` respectively.

Read the Manual about widget properties.

Parameters

- `k`: The widget property name.

`setProperty`(k, v)

General-purpose setter for widget properties.

See *getProperty()*.

Parameters

- `k`: The widget property name.
- `v`: The new value.

`bindProperty`(k, vb)

Binds a *DataBinding* to a property. Read Manual for details about data bindings.

Parameters

- `k`: The widget property name.
- `vb`: The *clove.DataBinding* to bind. May be 'undefined' for just unbinding.

`init`(rootNameScope)

Initializes the widget.

Never override this method in custom widgets. See *doresize()*.

Intended for usage by the infrastructure; never call this method directly.

Parameters

- `rootNameScope`: The root namescope to add this widget to.

`doinit`()

Executes late widget initialization (i.e. after properties are applied).

This is used for initialization steps which need the properties to be applied already. See also *doinitEarly()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`doinitEarly`()

Executes early widget initialization (i.e. before properties are applied).

This is used for initialization steps which need to run before property values are applied. See also *doinit()*.

Override this method in custom widgets.

Intended for usage by the infrastructure; never call this method directly.

`resize`()

Applies the new widget size to internal content.

Never override this method in custom widgets. See *doresize()*.

This function is typically called from the parent widget when the size has been changed.

doresize()

Corrects alignments of internal elements according to the new widget size.

Override this method in custom widgets.

Never call this method directly. See *resize()*.

relayout()

Notifies the parent widget that a new geometry is required.

This will eventually lead to a resizing according to *clove::Widget::getPreferredWidth* (et al).

This function is typically called from within the widget implementation when the content changed and from child widgets.

focus()

Sets the focus to this widget.

isAlive()

Checks if the widget is still alive (i.e. mounted somewhere in the dom tree).

computeMinimalWidth()

Computes the minimal width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computePreferredWidth()

Computes the preferred width in pixel this widget needs to have.

Override this method for geometry measurement in custom widgets.

computeMinimalHeightForWidth(w)

Computes the minimal height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

computePreferredHeightForWidth(w)

Computes the preferred height in pixel this widget needs to have for a given width.

Override this method for geometry measurement in custom widgets.

Parameters

- `w`: The width in pixel.

getMinimalWidth()

Returns the minimal width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getPreferredWidth()

Returns the preferred width in pixel this widget needs to have.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

getMinimalHeightForWidth(w)

Returns the minimal height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

getPreferredHeightForWidth(w)

Returns the preferred height in pixel this widget needs to have for a given width.

Do not override this method in custom widgets. This method caches geometry and does some other adjustments.

Parameters

- `w`: The width in pixel.

addStyleClass(clss)

Adds the css class `clss` to this widget.

Parameters

- `clss`: A css class name.

removeStyleClass(clss)

Removes the css class `clss` from this widget.

Parameters

- `clss`: A css class name.

isStyleClass(clss)

Checks if this widget contains the css class `clss`.

Parameters

- `clss`: A css class name.

setStyleClassAssigned(clss, assigned)

Sets or unsets the css class `clss` to this widget.

Parameters

- `clss`: A css class name.
- `assigned`: If to assign or unassign it.

containingNameScope()

The namescope this widget contains to.

nameScope()

The own namescope (or the namescope it contains to, if this widget does not bring a new one).

remove(removeconfig)

Removes this widget.

Note: This method does not guarantee to remove the widget synchronously, since there might be animations before.

Use *clove::Widget::OnDestroyed* for continuing after removal.

Parameters

- `removeconfig`: The removal configuration (optional and only for exotic cases).

effectivelyEnabled()

If this widget is enabled and not marked as busy (i.e. can interact with the user).

This checks if the parent widgets are enabled and non-busy as well. See also *enabled()* and *busy()*.

effectiveVisibility()

If this widget and all parent widgets are visible. See also *visibility()*.

childrenWidgets()

List of the children *clove::Widget* instances.

parentWidget()

The parent *clove::Widget*.

name()

The name of the widget.

This name can be used for getting the widget instance later on. Read the Manual about namescopes for details.

setName(value)

Setter for *name()*.

Parameters

- `value`: The new value.

enabled()

If this widget is marked as enabled (i.e. can interact with the user).

This does not check the parents. See also *effectivelyEnabled()*.

setEnabled(value)

Setter for *enabled()*.

Parameters

- `value`: The new value.

styleClass()

Custom css class(es).

setStyleClass(value)

Setter for *styleClass()*.

Parameters

- `value`: The new value.

style()

Custom css style string. You should not use that, but *styleClass()* instead.

setStyle(value)

Setter for *style()*.

Parameters

- `value`: The new value.

visibility()

If this widget is visible.

One of *clove::Visible*, *clove::Invisible* and *clove::InvisibleCollapsed*.

See also *effectiveVisibility()*.

setVisibility(value)

Setter for *visibility()*.

Parameters

- `value`: The new value.

doStandaloneResizing()

If the widget handles to resize itself as needed.

This is only needed in exotic situations and by the infrastructure.

setDoStandaloneResizing(value)

Setter for *doStandaloneResizing()*.

Parameters

- `value`: The new value.

mayFocus()

If the widget can have the keyboard focus.

setMayFocus(value)

Setter for *mayFocus()*.

Parameters

- `value`: The new value.

horizontalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional horizontal space.

setHorizontalStretchAffinity(value)

Setter for *horizontalStretchAffinity()*.

Parameters

- `value`: The new value.

verticalStretchAffinity()

Numeric value >= 0 which specifies how much this widget would benefit from additional vertical space.

setVerticalStretchAffinity(value)

Setter for *verticalStretchAffinity()*.

Parameters

- `value`: The new value.

strictHorizontalSizing()

If the widget is strictly forbidden to get additional horizontal space.

Otherwise there are situations where additional space is assigned even with `horizontalStretchAffinity()==0`.

`setStrictHorizontalSizing`(value)
Setter for *strictHorizontalSizing()*.

Parameters

- `value`: The new value.

`strictVerticalSizing`()
If the widget is strictly forbidden to get additional vertical space.

Otherwise there are situations where additional space is assigned even with `verticalStretchAffinity()==0`.

`setStrictVerticalSizing`(value)
Setter for *strictVerticalSizing()*.

Parameters

- `value`: The new value.

`vstretch`()
Alias for *verticalStretchAffinity()*.

`setVstretch`(value)
Setter for *vstretch()*.

Parameters

- `value`: The new value.

`hstretch`()
Alias for *horizontalStretchAffinity()*.

`setHstretch`(value)
Setter for *hstretch()*.

Parameters

- `value`: The new value.

`busy`()
If the widget is in busy state, typically resulting in a loading animation.

`setBusy`(value)
Setter for *busy()*.

Parameters

- `value`: The new value.

`registerBusy`()
Sets the widget to busy state and returns a token which helps returning to normal state.

See also *unregisterBusy()* and *busy()*.

You must not mix this function with direct usage of *setBusy()*!

`unregisterBusy`(token)
Drops a token and returns to normal state if no other tokens exist.

Parameters

- `token`: The token as returned by *registerBusy()*.

hasFocus()
If the widget has the keyboard focus.

This also returns true if a child has focus.

innerSize()
Returns inner width and height of this widget.

outerSize()
Returns outer width and height of this widget.

OnDestroyed
Triggered when this widget was removed somehow.

This is a *clove.Event* instance. See Manual for details.

OnResized
Triggered when this widget was resized.

This is a *clove.Event* instance. See Manual for details.

OnVisibilityChanged
Triggered when the visibility of this widget changed.

Only direct changes trigger the event, not when the visibility of a predecessor changed.

This is a *clove.Event* instance. See Manual for details.

OnPropertyChanged
Triggered whenever a property of this widget changed its value.

Note: A few properties are only computed on-demand and don't trigger this event.

This is a *clove.Event* instance. See Manual for details.

OnKeyDown
Triggered when a keyboard key went down.

This is a *clove.Event* instance. See Manual for details.

OnKeyUp
Triggered when a keyboard key came up.

This is a *clove.Event* instance. See Manual for details.

OnKeyPress
Triggered when a keyboard key was pressed.

Note that this event does not work for all keys in all browsers!

This is a *clove.Event* instance. See Manual for details.

children()
List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)
Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)
Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()
Removes all childs from the layout.

18.1.75 Class clove::WrapLayout

`class clove::WrapLayout : public clove::Layout`
A mixin for wrap layout implementations.

Subclassed by *clove::Wrap*

Public Functions

children()
List of all child widgets in this layout as widget configurations like in *clove::build()*.

setChildren(value)
Setter for *children()*.

Parameters

- `value`: The new value.

addChild(value)
Adds a new child widget to the layout.

Parameters

- `value`: The new widget as widget configuration like for *clove::build()*.

clearChilds()
Removes all childs from the layout.

18.1.76 Class clove::symbols

`class symbols`
Namespace for symbol constants.

They make a few Unicode symbols accessible by a name.

18.1.77 Class clove::utils

class *clove*::**utils**

Namespace for some utilities.

Public Functions

main_parts_resized(reason)

Call this method if resizing is required due to external situations.

It is not required to call this method in typical situations. A situation which requires this method is changing the stylesheets from outside.

Parameters

- `reason`: Optional arbitrary object which describes the reason for resizing. Can be evaluated by handlers in a custom way.

applyDefaultsToConfig(config, defaults)

Returns a copy of `config` with the additional members from `defaults`, whereever there was no value in `config`.

Parameters

- `config`: The source object.
- `defaults`: Default values for members to apply to the result whereever they are missing.

addOnDestroyHandler(domnode, handler)

Listen for the removal of a node in the dom tree and eventually execute a handler function.

Parameters

- `domnode`: The node in the dom tree to observe.
- `handler`: The function to execute on removal of `domnode`.

addOnDragHandler(domnode, dragbeginhandler, draghandler, dragendhandler)

Listen for a drag event of a node and eventually execute handler functions.

A drag consists of mouse/finger down, moving and releasing.

Parameters

- `domnode`: The node in the dom tree to observe.
- `dragbeginhandler`: The function to execute when a drag operation begins.
- `draghandler`: The function to execute while moving (with relative coordinates as parameters).
- `dragendhandler`: The function to execute when a drag operation ends.

addOnDoubleClickTapHandler(domnode, handler)

Listen for double click/tap event of a node and eventually execute a handler function.

This is like a pure double click event, but also supports touch.

See also *clove::utils::removeOnDoubleClickTapHandler()*.

Parameters

- `domnode`: The node in the dom tree to observe.

- `handler`: The function to execute on double click/tap.

`removeOnDoubleClickTapHandler`(domnode, handler)
Remove a double click/tap event handler.

Parameters

- `domnode`: The node in the dom tree to remove a handler from.
- `handler`: The function to remove.

`arraysum`(x)
Computes the sum of all elements in an Array.

Parameters

- `x`: The Array to sum up.

`findWidgetForDomNode`(node)
Finds the (most inner) *clove::Widget* a given dom node contains to.

Parameters

- `node`: The dom node.

`getExtraSize`(node, withMargin)
Computes the extra size of a dom node.

This is the difference between inner size and the outer size (i.e. borders, padding, …).

Parameters

- `node`: The dom node to measure.
- `withMargin`: If to include margins.

`getInnerSize`(node)
Computes the inner size of a dom node.

This is the size without borders, padding, …

Parameters

- `node`: The dom node to measure.

`getOuterSize`(node, withMargin)
Computes the outer size of a dom node.

This is the size including borders, padding, …

Parameters

- `node`: The dom node to measure.
- `withMargin`: If to include margins.

`suspendResizing`()
Temporarily suspends all user interface resizing.

This is useful for performance reasons during intensive changes to the user interface.

Use this with caution and always call *clove::utils::resumeResizing* afterwards.

Return A token object used for resuming later on.

resumeResizing (token)

Resumes the user interface resizing after it was suspended with *clove::utils::suspendResizing*.

It automatically takes care of resizing everything which diverged meanwhile.

Parameters

- `token`: The token object.

cssLengthToPixels (v, axis)

Translates a css length specification string to a numeric pixel value.

Note: This does not make sense for all kinds of lengths, e.g. percentages will not work.

Parameters

- `v`: The css length.
- `axis`: The axis ('x' or 'y').

insertToArray (array, i, v)

Inserts a value into an array.

Parameters

- `array`: The array.
- `i`: The position.
- `v`: The new value to insert.

popup (config, showconfig)

Shows a popup.

This can show an arbitrary widget decoupled from the main user interface (swimming above it).

This call retuns a dialog result structure, containing the following:

- `widget`: The popup widget as *clove::Widget*.
- `namescope`: The root namescope for the popup as *clove::NameScope*.

  Parameters

  - `config`: The widget configuration, as for *clove::build()*, which specifies the popup.
  - `showconfig`: A configuration object which specifies some presentation aspects.

isDescendantOf (d, p)

Checks if `d` is a descendant of `p` in the html dom node.

Parameters

- `d`: An html dom node.
- `p`: An html dom node.

arrayToDatasource (array)

Builds a *clove::Datasource* containing the same content as the input array.

Parameters

- `array`: The input array.

`connectDatasourceToWidget`(widget, datasource, dspropname, rootptrpropname, inserthandlername, removehandlername, updatehandlername, afterdisconnectedfct, beforeconnectedfct, afterconnectedfct)

Connects a *clove::Datasource* to an internal widget interface (also removes the old connection).

Connects a datasource to a widget.

This method is solely used by the inner implementation of a widget for using a datasource. It is not useful to call it in any other situation! Used internally in widget implementations.

Parameters

- `widget`: A *clove::Widget*.
- `datasource`: A *clove::Datasource*.
- `dspropname`: The property name for storing the current datasource in the widget.
- `rootptrpropname`: The property name to get the root *clove::DatasourceValuePointer* from the widget.
- `inserthandlername`: The name of the widget's insert handler method.
- `removehandlername`: The name of the widget's remove handler method.
- `updatehandlername`: The name of the widget's update handler method.
- `afterdisconnectedfct`: Optional function called just after disconnection of the old datasource.
- `beforeconnectedfct`: Optional function called just before connecting the new datasource.
- `afterconnectedfct`: Optional function called just after connecting the new datasource.

Parameters

- `widget`: The *clove::Widget*.
- `datasource`: The *clove::Datasource* to connect.
- `dspropname`: The name of the datasource property.
- `rootptrpropname`: The name of the root value pointer property.
- `inserthandlername`: The name of the insert handler.
- `removehandlername`: The name of the remove handler.
- `updatehandlername`: The name of the update handler.
- `afterdisconnectedfct`: This function is called after the last datasource is disconnected.
- `beforeconnectedfct`: This function is called just before the datasource is connected.
- `afterconnectedfct`: This function is called right after the datasource is connected.

`animateNode`(node, animationname, animationduration, animationendedfct)

Plays a css animation on a html dom node for one iteration and fires a callback afterwards.

Parameters

- `node`: The html dom node to animate.
- `animationname`: The css animation name.
- `animationduration`: The animation duration in milliseconds.
- `animationendedfct`: a callback `function()` called when the animation ends.

`setStyleClassAssigned`(node, clss, assigned)

Sets or unsets a css class to an html dom node.

Parameters

- `node`: The html dom node to modify.
- `clss`: The css class name.
- `assigned`: If to assign or unassign it.

`deferLoading`(eload)

Defers loading of Clove by an event.

Used only internally and in exotic situations.

Call this before loading is finished (i.e. typically while the document itself loads) and before the `eload` event triggers (i.e. not when it already might have triggered).

See also *runOnLoaded()*.

Parameters

- `eload`: The *clove::Event* object which is triggered when it's ready for continue loading.

`runOnLoaded`(fct)

Runs a function as soon as possible, but not before Clove has finished loading.

Used only internally and in exotic situations. One would typically use *clove::populateUI()* instead.

Parameters

- `fct`: The `function()` to execute when Clove is loaded.

`getFullPathForLoadedResource`(tgtname, elementtype, elementsrcattrname)

Returns the full path for an already loaded script or stylesheet with known file name.

Used only internally and in exotic situations.

Parameters

- `tgtname`: The complete file name (bare; without slashes) of the file to search for.
- `elementtype`: Optional element type (default: `"script"`).
- `elementsrcattrname`: Optional element type's attribute name for the source url (if type is not `"script"` or `"style"`).

`tryLoadScript`(scriptpath, config)

Tries to load another javascript.

If it succeeds, it returns the url of it.

config may contain:

- `rootrefscript`: For relative script paths, this reference script is used in order to determine the base directory.
- `onload`: A `function()` to execute when the script is loaded.

  Parameters

  - `scriptpath`: The path to the javascript file.
  - `config`: A (optional) configuration object.

`tryLoadStyle`(stylepath, config)

Tries to load a css stylesheet.

If it succeeds, it returns the url of it.

config may contain:

- `rootrefscript`: For relative style paths, this reference script is used in order to determine the base directory.
- `onload`: A `function()` to execute when the stylesheet is loaded.

  Parameters

  - `stylepath`: The path to the css stylesheet file.
  - `config`: A (optional) configuration object.

`OnMainResized`

Triggered when the main view resized.

This is a *clove.Event* instance. See Manual for details.

INDEX