Sweet Home 3D

Sweet Home 3D Online - User guide

Version 1.0

Renaud Pawlak
renaud.pawlak@gmail.com / renaud.pawlak@cincheo.fr
http://www.sh3d.online

Contents

Introduction

Sweet Home 3D™ is a 2D/3D floor planning and plan editor/viewer Open Source software. It has been developed by Emmanuel Puybaret (eTeks) for the last 15 years and has a strong and active community.

Sweet Home 3D JS is also Open Source and is the Web version of Sweet Home 3D. It has been co-developed by Emmanuel Puybaret (eTeks) and Renaud Pawlak (Cinchéo). A demo version can be found at `http://www.sweethome3d.com/SweetHome3DOnlineManager.jsp`.

The Sweet Home 3D Online™ product (SH3DO™ for short) allows businesses to take advantage of Sweet Home 3D and Sweet Home 3D JS. It provides a ready-to-use server and an Web-Service-based API for easy integration in any web site. The constraints of the Open Source licensing are relaxed so that one can fully customize the product for its own constraints and needs.

SH3DO™ entails with the following characteristics and components:

- 2D plan editor for the Web;
- 3D viewer (in sync with the 2D plan editor);
- furniture catalog web component;
- compatibility with Safari, Chrome and IE, including mobile versions;
- easy ecommerce website integration (pure HTML/Javascript, frameworkless);
- cloud servers for serving the viewer component + reading and saving project files (for development, pre-production, and production environments);
- maintenance;
- backups;
- an integration and customization API (brand, colors, layout, etc);
- tools for furniture / object catalog customization.

This document aims at providing the technical documentation to integrate and customize SH3DO™ in your own Web site. It requires basic Web programming knowledge. If you do not have programming knowledge in house, Cinchéo provides development services in order to help you with SH3DO™ integration and customization.

1. Getting started

1.1 Registration and Retailer Keys

The commercial licensing of Sweet Home 3D Online is brought to you by Cinchéo / eTeks. This license allows you to tune the Sweet Home 3D Online product and integrate it in a website. Once you have purchased a license, you will get a Retailer Key that will allow you to access one of the available servers (environments).

For each environment, a Retailer Key is bound to a Domain Name (for instance `https://mycompany.com`), which is the domain that host the web site you want to integrate SH3DO with. Any request coming from another origin than the declared domain name will not be allowed.

Note that the communication between your web site and the SH3DO services is secured with CORS and HTTPS.

1.2 Environments (dev, preprod, prod)

Your Retailer Key will allow you to securely access the following environments:

- Your development environment (`https://dev.sh3d.online`). This environment will accept requests with your Retailer Key coming only from `http://localhost:8000` or `http://localhost:8080`. It is to be used for development and debugging purposes only. Data and files may be erased at any time and should be stored for testing purpose.
- Your pre-production/QA environment (`https://preprod.sh3d.online`). This optional environment will accept requests with your Retailer Key coming from the origin domain you will have declared to us. It is to be used for QA and testing purposes only. Data and files may be erased at any time and should be stored for testing purpose.
- Your production environment (`https://prod.sh3d.online`). This is the actual environment, which will accept requests with your Retailer Key coming from the origin domain you will have declared to us. User data will be safely backed-up and will remain there as long as you need it and hold a valid license.

1.3 The Demo environment

For testing the API without having your Retailer Key available yet, you may access the demo environment, which is open to everyone and can be used with the Retailer Key 12345.

The following code shows a typical example to show a full SH3DO plan editor / 3D view / Furniture catalog using the 12345 Retailer Key on the demo server. In order to make it work, you need to start a local HTTP server to serve the file.

Assuming that you have Python installed on your computer, one of the simplest way is to use a Python HTTP server. Follow the steps:

- Create an `index.html` file and copy the HTML content below in the file.
- Open a terminal, go to the folder where you just have created the index file.
- Launch the HTTP server with the command: `python -m SimpleHTTPServer`.
- Open the `http://localhost:8000` with your favorite browser.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="minimal-ui, user-scalable=no, initial-scale=1,
    maximum-scale=1, minimum-scale=1, width=device-width">
<meta name="format-detection" content="telephone=no">
<meta name="msapplication-tap-highlight" content="no">
<title>SweetHome3DJS Test</title>
<script type="text/javascript"
    src="https://demo.sh3d.online/lib/sweethome3d.min.js"></script>
<link rel="stylesheet" type="text/css"
    href="https://demo.sh3d.online/css/sweethome3d.css">
</head>
<body>

<div id="home-pane">
  <canvas id="home-3D-view" style="background-color: #CCCCCC;"
        tabIndex="1"></canvas>
  <div id="home-plan" style="background-color: #FFFFFF; color: #000000"
        tabIndex="2" ></div>
  <div id="home-pane-toolbar"></div>
  <div id="furniture-catalog"></div>
</div>

<script type="text/javascript">
var homeName = "HomeTest";
var urlBase = "https://demo.sh3d.online/";
var application = new SweetHome3DJSApplication(
   {readHomeURL:  urlBase + "api/readHome/12345/test?home=%s",
    writeHomeEditsURL: urlBase + "api/writeHomeEdits/12345/test?home=%s",
    //closeHomeURL: urlBase + "closeHome.jsp?home=%s",
    pingURL:      urlBase + "api/ping",
    furnitureCatalogURLs: [urlBase +
        "lib/resources/DefaultFurnitureCatalog.json"],
    furnitureResourcesURLBase: urlBase,
    autoWriteDelay: 1000,
    autoWriteTrackedStateChange: true,
    writingObserver: {
       writeStarted: function(update) {
         console.info("Update started", update);
       },
       writeSucceeded: function(update) {
         console.info("Update succeeded", update);
       },
       writeFailed: function(update, errorStatus, errorText) {
         console.info("Update failed", update);
       },
       connectionFound: function() {
         console.info("Back to online mode");
       },
       connectionLost: function(errorStatus, errorText) {
         console.info("Lost server connection - going offline");
       }
     }
   });

// Set a few settings
application.getUserPreferences().setNewRoomFloorColor(0xFF9999A0);
```

```
application.getUserPreferences().setAerialViewCenteredOnSelectionEnabled(true);

// Read and display test file
application.getHomeRecorder().readHome(homeName,
   {
     homeLoaded: function(home) {
       home.setName(homeName);
       application.addHome(home);

       window.addEventListener("unload", function() {
          application.deleteHome(home);
         });
     },
     homeError: function(err) {
       console.error(err);
     },
     progression: function(part, info, percentage) {
     }
   });
</script>

</body>
</html>
```

Listing 1.1: Basic example

2. Integration and Customization examples

One can integrate SH3DO with any web site. Integration requires the following:

- Use the REST API to a SH3DO server to connect, create users, create and save plans, and so on;
- Use the JavaScript API to create the components and customize the behavior;
- Use HTML and CSS to customize the page layout, tune the Look and Feel, and integrate the SH3DO components in your web site.

2.1 How to customize the layout

The layout is fully defined in the `lib/sweethome3d.css` stylesheet. If you want to tune the default layout, you need to understand the structure of this file, then create your own CSS rules. You can place the new CSS in the header of your HTML file, or create a new .css file and link it after `lib/sweethome3d.css`.

As shown in listing 1.1, each SweetHome3D component has an id. So, the CSS rule will apply to these:

```
[...]
<div id="home-pane">
  <canvas id="home-3D-view" tabIndex="1"></canvas>
  <div id="home-plan" tabIndex="2" ></div>
  <div id="home-pane-toolbar"></div>
  <div id="furniture-catalog"></div>
</div>
[...]
```

As an example, the following CSS will remove the 3D view (id: home-3D-view) from the page and make the plan view (id:home-plan) full page:

```
#home-3D-view {
  display: none
}

#home-plan {
  height: calc(100% - 16px);
}
```

2.2 How to change the default colors

Similarly to the layout, color can be changed tuning the `lib/sweethome3d.css` stylesheet. Be aware that CSS rule can be overridden, especially by local declaration directly in the HTML style attribute of the elements.

For instance, in order to invert the colors of the plan view (black background and white foreground), you can change the styling of the plan view HTML element like this:

```
  <div id="home-plan" style="background-color: black; color: white"
       tabIndex="2" ></div>
```

You can also put it in a CSS file:

```
#home-plan {
  background-color: black;
  color: white;
}
```

2.3 How to change the default user preferences

You can access the user preferences with JavaScript. The entry point is the application variable accessible from the root scope (the HTML page).

For example, you can force the 3D aerial view to be centered around the selection with the following JavaScript line to be added after the creation of the application object:

```
  application.getUserPreferences()
    .setAerialViewCenteredOnSelectionEnabled(true);
```

The full API for the user preferences is available at: `http://www.sweethome3d.com/javadoc/com/eteks/sweethome3d/model/UserPreferences.html`.

2.4 How to add a custom button to the toolbar

There are several ways to add a new button to the toolbar. A simple way to do it is through the following JavaScript function.

```
function addButton(id, title, imageUrl, onclick) {
   var toolbar = document.getElementById('home-pane-toolbar');
   toolbar.insertAdjacentHTML('beforeend', "<span
       class='toolbar-button-group'><button title='"+title+"'
       class='toolbar-button toggle selected' id='"+id+"'
       style='background-image: url(&quot;"+imageUrl+"&quot;);
       background-position: center center; background-repeat:
       no-repeat;background-color:#fff'></button></span>");
   document.getElementById(id).addEventListener('click', onclick);
}
```

You can call this function once the home is created, in the "homeLoaded" handler of the application:

```
[...]
application.getHomeRecorder().readHome(homeName,
  {
    homeLoaded: function(home) {
      home.setName(homeName);
      application.addHome(home);

      addButton(
        'custom-button-id',
        'my new button',
        'http://demo.sh3d.online/lib/search.gif',
        function() {
          alert("hello from my new button");
        });
[...]
```

3. API

3.1 General-purpose REST API

3.1.1 Endpoint `ping`

This endpoint may be used by the client to check the server availability.

Path	`ping`
HTTP method	GET
Parameters	-
Result	status object (JSON)

3.1.2 Endpoint `getRetailer`

This endpoint gets the retailer information for the given retailer key. Keys are provided to you by CINCHEO. Please contact Renaud Pawlak to get your own key.

Path	`getRetailer/retailerKey`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key
Result	A JSON object containin the following data: • `domain`: the retailer's web site domain • `email`: the retailer's contact email • `key`: the retailer's key • `enabled`: a flag telling if the retailer's account is activated • `plan`: the name of the plan (DEMO | STARTER | BASIC | PROFESSIONAL | PREMIUM) • `planExtensionCode1`: an optional integer encoding an extension for the retailer's plan • `planExtensionCode2`: an optional integer encoding an extension for the retailer's plan

3.1.3 Endpoint `getRetailerStats`

This endpoint gets the retailer statistics for the given retailer key.

Path	`getRetailerStats/retailerKey`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key
Result	A JSON object containin the following data: • `lastReadHome`: the retailer's web site domain • `userCount`: the total number of users for this retailer • `homeCount`: the total number of homes (plans) for this retailer • `usedSpace`: the total space used by this retailer's homes (in bytes) • `lastReadHome`: the date when the last readHome access occurred • `readHomeCount`: the total count of readHome access since the beginning of the plan • `dailyReadHomeCount`: the total count of readHome access since the beginning of the current day (or since the lastReadHome date if no accesses) • `monthlyReadHomeCount`: the total count of readHome access since the beginning of the current month (or since the lastReadHome date if no accesses) • `yearlyReadHomeCount`: the total count of readHome access since the beginning of the current year (or since the lastReadHome date if no accesses) • `lastWriteHome`: the date when the last readHome access occurred • `writeHomeCount`: the total count of writeHome access since the beginning of the plan • `writeReadHomeCount`: the total count of writeHome access since the beginning of the current day (or since the lastWriteHome date if no accesses) • `monthlyWriteHomeCount`: the total count of writeHome access since the beginning of the current month (or since the lastWriteHome date if no accesses) • `yearlyWriteHomeCount`: the total count of writeHome access since the beginning of the current year (or since the lastWriteHome date if no accesses)

3.2 User-management REST API

Retailers can register and manage users. Without users (or at least a default user) registered on the server, retailers cannot create new homes (plans).

3.2.1 Endpoint `registerUser`

This endpoint creates a new user for a given retailer on the server. Once created, a use will have a space on the server to create new homes (see createNewHome).

Path	`registerUser/retailerKey/userId`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the new user id (must not exist yet)
Result	status object (JSON)

3.2.2 Endpoint `getUsers`

This endpoint returns all the user ids (list of strings) that have been registered for this retailer (see registerUser).

Path	`getUsers/retailerKey`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key
Result	a list of user ids as JSON

3.2.3 Endpoint `deleteUser`

This endpoint creates a new user for a given retailer on the server. Once created, a user will have a space on the server to create new homes (see createNewHome and uploadHome).

Path	`deleteUser/retailerKey/userId`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the new user id (must exist for the given retailer)
Result	status object (JSON)

3.3 Home-management REST API

Homes are SH3D files that are created for each user registered on the server. A home is a plan that can be edited (and saved if the plan allows it) and visualized in 3D.

3.3.1 Endpoint `createNewHome`

This endpoint creates a new (empty) home for the given user.

Path	`createNewHome/retailerKey/userId/homeName`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the user id, belonging to the retailer, as provided to registerUser • `homeName` (path parameter): the home name (should not contain special characters or accents to avoid potential Locale issues)
Result	status object (JSON)
Errors	• `IOException`: if home already exists of if a problem occurs while creating the new home file on the server • `RecorderException`: if Sweet Home 3D recorder fails at creating a new empty home

3.3.2 Endpoint `uploadHome`

This endpoint creates a new home for the given user by uploading an SH3D file.

Path	`uploadHome/retailerKey/userId`
HTTP method	POST
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the user id, belonging to the retailer, as provided to registerUser • `homeName` (form parameter): the home name (should not contain special characters or accents to avoid potential Locale issues) • `file` (multi-part file form parameter): the home file as provided to the form file upload
Result	status object (JSON)
Errors	• `IOException`: if home already exists of if a problem occurs while creating the new home file on the server • `RecorderException`: if Sweet Home 3D recorder fails at creating a new empty home

3.3.3 Endpoint `getHomes`

This endpoint gets all the created homes (plans) for the given user.

Path	`getHomes/retailerKey/userId`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the user id, belonging to the retailer, as provided to registerUser
Result	a list of home names (string) as JSON

3.3.4 Endpoint `readHome`

This endpoint reads the given home on the server and downloads it in SH3D file format. It is to be used by a Sweet Home 3D JavaScript client.

This endpoint comes with a POST and a GET version.

Path	`readHome/retailerKey/userId/home`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the user id, belonging to the retailer, as provided to registerUser • `home` (path parameter): the home name, as provided to `createNewHome`
Result	a byte stream containing the home file in binary format
Errors	• `IOException`: if home does not exist or if a system error occurs while reading the file

Path	`writeHomeEdits/retailerKey/userId`
HTTP method	POST
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the user id, belonging to the retailer, as provided to registerUser • `homeName` (form/header parameter): the home name, as provided to `readHome` • `edits` (form/header parameter): the list of edits to apply to the home
Result	a result object containing the count of applied edits (JSON)
Errors	• `Throwable`: if any of the parameter is wrong or if the edits are not consistent with the target home

3.3.5 Endpoint `deleteHome`

This endpoint permanently deletes a home from the server.

Path	`deleteHome/retailerKey/userId/home`
HTTP method	GET
Parameters	• `retailerKey` (path parameter): the retailer key • `userId` (path parameter): the user id, belonging to the retailer, as provided to registerUser • `home` (path parameter): the home name, as provided to `createNewHome`
Result	a byte stream containing the home file in binary format
Errors	• `IOException`: if home does not exist or if a system error occurs while reading the file

3.4 JavaScript API

Although there are minor differences and that some elements are not fully supported, the JavaScript API mainly conforms to the Java API of SweetHome3D. Please note that the model and controller packages are automatically generated from Java with the JSweet Java to JavaScript compiler.

In order to navigate the API, go to `http://www.sweethome3d.com/javadoc/`. This API will give you a good starting point to access the application and the underlying home model, and dynamically interact with is in JavaScript (see the com.eteks.sweethome3d.controller and com.eteks.sweethome3d.model packages).

Additionally, you can refer to the following APIs, which have been generated with jsdoc from the JavaScript source file of SweetHome3DJS.

3.4.1 Classes

IncrementalHomeRecorder
SweetHome3DJSApplication
PlanComponent

3.4.2 IncrementalHomeRecorder

Kind: global class
Author: Renaud Pawlak
Author: Emmanuel Puybaret

- IncrementalHomeRecorder
  - new IncrementalHomeRecorder(application, [configuration])
  - *instance*
    - .getTrackedHomeProperties()
    - .configure(configuration)
    - .readHome(homeName, observer)
    - .addHome(home)
    - .removeHome(home)
    - .sendUndoableEdits(home)
  - *static*
    - .DEFAULT_TRACKED_HOME_PROPERTIES

new IncrementalHomeRecorder(application, [configuration])

A home recorder that is able to save the application homes incrementally by sending undoable edits to the SH3D server.

Param	Type	Description
application	SweetHome3DJSApplication	
[configuration]	Object	the recorder configuration

incrementalHomeRecorder.getTrackedHomeProperties()

Gets the home properties that are tracked and potentially written by this recorder.
Kind: instance method of IncrementalHomeRecorder

incrementalHomeRecorder.configure(configuration)

Configures this incremental recorder's behavior.
Kind: instance method of IncrementalHomeRecorder

Param	Type	Description
configuration	Object	an object containing configuration fields

incrementalHomeRecorder.readHome(homeName, observer)

Reads a home with this recorder.
Kind: instance method of IncrementalHomeRecorder

Param	Type	Description
homeName	string	the home name on the server or the URL of the home if readHomeURL service is missing
observer	Object	callbacks used to follow the reading of the home

incrementalHomeRecorder.addHome(home)

Adds a home to be incrementally saved by this recorder.
Kind: instance method of IncrementalHomeRecorder

Param	Type
home	Home

incrementalHomeRecorder.removeHome(home)

Removes a home previously added with addHome.
Kind: instance method of IncrementalHomeRecorder

Param	Type
home	Home

incrementalHomeRecorder.sendUndoableEdits(home)

Sends to the server all the currently waiting edits for the given home.
Kind: instance method of IncrementalHomeRecorder

Param	Type
home	Home

IncrementalHomeRecorder.DEFAULT_TRACKED_HOME_PROPERTIES

The home properties that are tracked by default by incremental recorders.
Kind: static property of IncrementalHomeRecorder

3.4.3 SweetHome3DJSApplication

Kind: global class
Author: Emmanuel Puybaret
Author: Renaud Pawlak

new SweetHome3DJSApplication([params])

Defines HomeApplication implementation for JavaScript.

Param	Type	Description
[params]	Object	the URLs of resources and services required on server (if undefined, will use local files for testing)

3.4.4 PlanComponent

Kind: global class
Author: Emmanuel Puybaret
Author: Renaud Pawlak

- PlanComponent
  - new PlanComponent(containerOrCanvasId, home, preferences, [object3dFactory], controller)
  - *instance*
    - .getHTMLElement()

    * .getPreferredSize() : java.awt.Dimension
    * .getPaintedItems() : Array.<Object>
    * .getItemBounds(g, item) : java.awt.geom.Rectangle2D
    * .getTextBounds(text, style, x, y, angle) : Array
    * .getFont([defaultFont], [textStyle]) : string
    * .getFontMetrics(defaultFont, textStyle) : FontMetrics
    * .setBackgroundPainted(backgroundPainted)
    * .isBackgroundPainted() : boolean
    * .setSelectedItemsOutlinePainted(selectedItemsOutlinePainted)
    * .isSelectedItemsOutlinePainted() : boolean
    * .repaint()
    * .getGraphics() : Graphics2D
    * .paintComponent(g)
    * .getPrintPreferredScale(g, pageFormat) : number
    * .createTransferData(dataType) : Object
    * .getClipboardImage() : HTMLImageElement
    * .isFormatTypeSupported(formatType) : boolean
    * .exportData(out, formatType, settings)
    * .getForegroundColor(mode) : string
    * .getBackgroundColor(mode) : string
    * .paintHomeItems(g, planScale, backgroundColor, foregroundColor, paintMode)
    * .getSelectionColor() : string
    * .getFurnitureOutlineColor() : string
    * .getIndicator(item, indicatorType) : Object
    * .isViewableAtSelectedLevel(item) : boolean
    * .setRectangleFeedback(x0, y0, x1, y1)
    * .makeSelectionVisible()
    * .makePointVisible(x, y)
    * .moveView(dx, dy)
    * .getScale() : number
    * .setScale(scale)
    * .convertXPixelToModel(x) : number
    * .convertYPixelToModel(y) : number
    * .convertXModelToScreen(x) : number
    * .convertYModelToScreen(y) : number
    * .getPixelLength() : number
    * .setCursor(cursorType)
    * .setToolTipFeedback(toolTipFeedback)
    * .setToolTipEditedProperties(toolTipEditedProperties, toolTipPropertyValues, x, y)
    * .deleteToolTipFeedback()
    * .setResizeIndicatorVisible(resizeIndicatorVisible)
    * .setAlignmentFeedback(alignedObjectClass, alignedObject, x, y, showPointFeedback)
    * .setAngleFeedback(xCenter, yCenter, x1, y1, x2, y2)
    * .setDraggedItemsFeedback(draggedItems)
    * .setDimensionLinesFeedback(dimensionLines)
    * .deleteFeedback()
    * .canImportDraggedItems(items, x, y) : boolean
    * .getPieceOfFurnitureSizeInPlan(piece) : Array
    * .isFurnitureSizeInPlanSupported() : boolean
    * .getHorizontalRuler() : Object
    * .getVerticalRuler() : Object
- *static*
    * .IndicatorType
        · new PlanComponent.IndicatorType()
        · .name() : string

- .toString() : string

* .PaintMode
* .getDefaultSelectionColor(planComponent) : string

new PlanComponent(containerOrCanvasId, home, preferences, [object3dFactory], controller)

Creates a new plan that displays home.

Param	Type	Description
containerOrCanvasId	string	the ID of a HTML DIV or CANVAS
home	Home	the home to display
preferences	UserPreferences	user preferences to retrieve used unit, grid visibility...
[object3dFactory]	Object	a factory able to create 3D objects from home furniture. The [Selectable, boolean) createObject3D](Object3DFactory#createObject3D(Home,) of this factory is expected to return an instance of `Object3DBranch` in current implementation.
controller	PlanController	the optional controller used to manage home items modification

planComponent.getHTMLElement()

Returns the HTML element used to view this component at screen.
Kind: instance method of PlanComponent

planComponent.getPreferredSize() : java.awt.Dimension

Returns the preferred size of this component in actual screen pixels size.
Kind: instance method of PlanComponent

planComponent.getPaintedItems() : Array.<Object>

Returns the collection of walls, furniture, rooms and dimension lines of the home painted by this component wherever the level they belong to is selected or not.
Kind: instance method of PlanComponent

planComponent.getItemBounds(g, item) : java.awt.geom.Rectangle2D

Returns the bounds of the given item.
Kind: instance method of PlanComponent

Param	Type
g	Graphics2D
item	Object

planComponent.getTextBounds(text, style, x, y, angle) : Array

Returns the coordinates of the bounding rectangle of the text centered at the point (x,y).
Kind: instance method of PlanComponent

Param	Type
text	string
style	TextStyle
x	number
y	number
angle	number

planComponent.getFont([defaultFont], [textStyle]) : string

Returns the HTML font matching a given text style.
Kind: instance method of PlanComponent

Param	Type
[defaultFont]	string
[textStyle]	TextStyle

planComponent.getFontMetrics(defaultFont, textStyle) : FontMetrics

Returns the font metrics matching a given text style.
Kind: instance method of PlanComponent

Param	Type
defaultFont	string
textStyle	TextStyle

planComponent.setBackgroundPainted(backgroundPainted)

Sets whether plan's background should be painted or not. Background may include grid and an image.
Kind: instance method of PlanComponent

Param	Type
backgroundPainted	boolean

planComponent.isBackgroundPainted() : boolean

Returns true if plan's background should be painted.
Kind: instance method of PlanComponent

planComponent.setSelectedItemsOutlinePainted(selectedItemsOutlinePainted)

Sets whether the outline of home selected items should be painted or not.
Kind: instance method of PlanComponent

Param	Type
selectedItemsOutlinePainted	boolean

planComponent.isSelectedItemsOutlinePainted() : boolean

Returns true if the outline of home selected items should be painted.
Kind: instance method of PlanComponent

planComponent.repaint()

Repaints this component elements when possible.
Kind: instance method of PlanComponent

planComponent.getGraphics() : Graphics2D

Returns a Graphics2D object.
Kind: instance method of PlanComponent

planComponent.paintComponent(g)

Paints this component.
Kind: instance method of PlanComponent

Param	Type
g	Graphics2D

planComponent.getPrintPreferredScale(g, pageFormat) : number

Returns the print preferred scale of the plan drawn in this component to make it fill pageFormat imageable size.
Kind: instance method of PlanComponent

Param	Type
g	Graphics2D
pageFormat	java.awt.print.PageFormat

planComponent.createTransferData(dataType) : Object

Returns an image of selected items in plan for transfer purpose.
Kind: instance method of PlanComponent

Param	Type
dataType	TransferableView.DataType

planComponent.getClipboardImage() : HTMLImageElement

Returns an image of the selected items displayed by this component (camera excepted) with no outline at scale 1/1 (1 pixel = 1cm).
Kind: instance method of PlanComponent

planComponent.isFormatTypeSupported(formatType) : boolean

Returns true if the given format is SVG.
Kind: instance method of PlanComponent

Param	Type
formatType	ExportableView.FormatType

planComponent.exportData(out, formatType, settings)

Writes this plan in the given output stream at SVG (Scalable Vector Graphics) format if this is the requested format.

Kind: instance method of PlanComponent

Param	Type
out	java.io.OutputStream
formatType	ExportableView.FormatType
settings	Object

planComponent.getForegroundColor(mode) : string

Returns the foreground color used to draw content.

Kind: instance method of PlanComponent

Param	Type
mode	PaintMode

planComponent.getBackgroundColor(mode) : string

Returns the background color used to draw content.

Kind: instance method of PlanComponent

Param	Type
mode	PaintMode

planComponent.paintHomeItems(g, planScale, backgroundColor, foregroundColor, paintMode)

Paints home items at the given scale, and with background and foreground colors. Outline around selected items will be painted only under PAINT mode.

Kind: instance method of PlanComponent

Param	Type
g	Graphics2D
planScale	number
backgroundColor	string
foregroundColor	string
paintMode	PaintMode

planComponent.getSelectionColor() : string

Returns the color used to draw selection outlines.

Kind: instance method of PlanComponent

planComponent.getFurnitureOutlineColor() : string

Returns the color used to draw furniture outline of the shape where a user can click to select a piece of furniture.

Kind: instance method of PlanComponent

planComponent.getIndicator(item, indicatorType) : Object

Returns the shape of the given indicator type.

Kind: instance method of PlanComponent

Param	Type
item	Object
indicatorType	IndicatorType

planComponent.isViewableAtSelectedLevel(item) : boolean

Returns true if the given item can be viewed in the plan at the selected level.

Kind: instance method of PlanComponent

Param	Type
item	Object

planComponent.setRectangleFeedback(x0, y0, x1, y1)

Sets rectangle selection feedback coordinates.

Kind: instance method of PlanComponent

Param	Type
x0	number
y0	number
x1	number
y1	number

planComponent.makeSelectionVisible()

Ensures selected items are visible at screen and moves scroll bars if needed.

Kind: instance method of PlanComponent

planComponent.makePointVisible(x, y)

Ensures the point at (x, y) is visible, moving scroll bars if needed.

Kind: instance method of PlanComponent

Param	Type
x	number
y	number

planComponent.moveView(dx, dy)

Moves the view from (dx, dy) unit in the scrolling zone it belongs to.

Kind: instance method of PlanComponent

Param	Type
dx	number
dy	number

planComponent.getScale() : number

Returns the scale used to display the plan.
Kind: instance method of PlanComponent

planComponent.setScale(scale)

Sets the scale used to display the plan. If this component is displayed in a scrolled panel the view position is updated to ensure the center's view will remain the same after the scale change.
Kind: instance method of PlanComponent

Param	Type
scale	number

planComponent.convertXPixelToModel(x) : number

Returns x converted in model coordinates space.
Kind: instance method of PlanComponent

Param	Type
x	number

planComponent.convertYPixelToModel(y) : number

Returns y converted in model coordinates space.
Kind: instance method of PlanComponent

Param	Type
y	number

planComponent.convertXModelToScreen(x) : number

Returns x converted in screen coordinates space.
Kind: instance method of PlanComponent

Param	Type
x	number

planComponent.convertYModelToScreen(y) : number

Returns y converted in screen coordinates space.
Kind: instance method of PlanComponent

Param	Type
y	number

planComponent.getPixelLength() : number

Returns the length in centimeters of a pixel with the current scale.
Kind: instance method of PlanComponent

planComponent.setCursor(cursorType)

Sets the cursor of this component.
Kind: instance method of PlanComponent

Param	Type
cursorType	PlanView.CursorType | string

planComponent.setToolTipFeedback(toolTipFeedback)

Sets tool tip text displayed as feedback.
Kind: instance method of PlanComponent

Param	Type	Description
toolTipFeedback	string	the text displayed in the tool tip or null to make tool tip disappear.

planComponent.setToolTipEditedProperties(toolTipEditedProperties, toolTipPropertyValues, x, y)

Set tool tip edition.
Kind: instance method of PlanComponent

Param	Type
toolTipEditedProperties	Array
toolTipPropertyValues	Array
x	number
y	number

planComponent.deleteToolTipFeedback()

Deletes tool tip text from screen.
Kind: instance method of PlanComponent

planComponent.setResizeIndicatorVisible(resizeIndicatorVisible)

Sets whether the resize indicator of selected wall or piece of furniture should be visible or not.
Kind: instance method of PlanComponent

Param	Type
resizeIndicatorVisible	boolean

planComponent.setAlignmentFeedback(alignedObjectClass, alignedObject, x, y, showPoint-Feedback)

Sets the location point for alignment feedback.

Kind: instance method of PlanComponent

Param	Type
alignedObjectClass	Object
alignedObject	Object
x	number
y	number
showPointFeedback	boolean

planComponent.setAngleFeedback(xCenter, yCenter, x1, y1, x2, y2)

Sets the points used to draw an angle in plan view.
Kind: instance method of PlanComponent

Param	Type
xCenter	number
yCenter	number
x1	number
y1	number
x2	number
y2	number

planComponent.setDraggedItemsFeedback(draggedItems)

Sets the feedback of dragged items drawn during a drag and drop operation, initiated from outside of plan view.
Kind: instance method of PlanComponent

Param	Type
draggedItems	Array.<Object>

planComponent.setDimensionLinesFeedback(dimensionLines)

Sets the given dimension lines to be drawn as feedback.
Kind: instance method of PlanComponent

Param	Type
dimensionLines	Array.<DimensionLine>

planComponent.deleteFeedback()

Deletes all elements shown as feedback.
Kind: instance method of PlanComponent

planComponent.canImportDraggedItems(items, x, y) : boolean

Returns true.
Kind: instance method of PlanComponent

Param	Type
items	Array.<Object>
x	number

Param	Type
y	number

planComponent.getPieceOfFurnitureSizeInPlan(piece) : Array

Returns the size of the given piece of furniture in the horizontal plan, or null if the view isn't able to compute such a value.
Kind: instance method of PlanComponent

Param	Type
piece	HomePieceOfFurniture

planComponent.isFurnitureSizeInPlanSupported() : boolean

Returns true if this component is able to compute the size of horizontally rotated furniture.
Kind: instance method of PlanComponent

planComponent.getHorizontalRuler() : Object

Returns the component used as an horizontal ruler for this plan.
Kind: instance method of PlanComponent

planComponent.getVerticalRuler() : Object

Returns the component used as a vertical ruler for this plan.
Kind: instance method of PlanComponent

PlanComponent.IndicatorType

Kind: static class of PlanComponent

- .IndicatorType
  - new PlanComponent.IndicatorType()
  - .name() : string
  - .toString() string

new PlanComponent.IndicatorType() Indicator types that may be displayed on selected items.

indicatorType.name() : string **Kind**: instance method of IndicatorType

indicatorType.toString() : string **Kind**: instance method of IndicatorType

PlanComponent.PaintMode

The circumstances under which the home items displayed by this component will be painted.
Kind: static enum of PlanComponent
Properties

Name	Type
PAINT	PaintMode

Name	Type
PRINT	PaintMode
CLIPBOARD	PaintMode
EXPORT	PaintMode

PlanComponent.getDefaultSelectionColor(planComponent) string

Returns the default color used to draw selection outlines. Note that the default selection color may be forced using CSS by setting backgroundColor with the ::selection selector. In case the browser does not support the ::selection selector, one can force the selectio color in JavaScript by setting PlanComponent.DEFAULT_SELECTION_COLOR.

Kind: static method of PlanComponent

Param	Type
planComponent	PlanComponent