



Version 1.0.1

Introduction	3
Mini-roadmap	3
Installation	3
The OpenSpace scene	4
The map elements	5
Tiles	5
Skins	6
Placing large 2D elements in the 3D space	8
Background and foreground	13
Using the Editor	15
Main menu	16
Skin Editor	18
Tile Editor	19
Supertile Editor	22
Background Editor	24
Map Editor	25



Version 1.0.1

Introduction

OpenSpace is an advanced isometric world engine which leverages the power of ActionScript 3 and SmartFoxServer to build unique MMO worlds, offering an unprecedented level of features and customizations.

The OpenSpace Editor is a tool designed to create the virtual environments that the OpenSpace Engine will then render and make interactive.

The OpenSpace Editor features:

- multiple editor projects management (create, open, save);
- world global configuration (tile size, tile aspect ratio, etc.);
- skins library management, with auto skin generation from swf file (Skin Editor);
- tiles editing and library management (Tile Editor);
- supertiles editing and library management (Supertile Editor);
- backgrounds and foregrounds editing and library management (Background Editor);
- maps editing and library management (Map Editor);
- map xml file exporting for OpenSpace Engine usage.

Some of the functionalities of the Editor are ready for the next version of the OpenSpace Engine, and are not applicable to the current version. A specific warning has been added in the following chapters where needed.

Installation

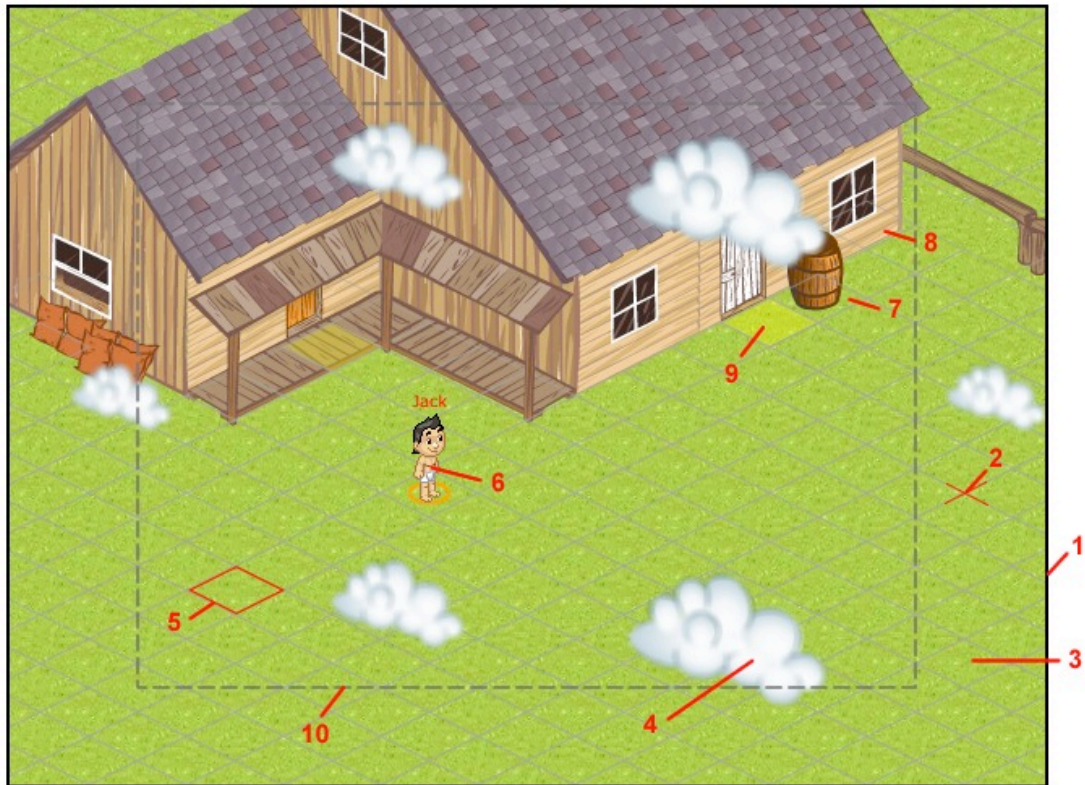
In order to install the OpenSpace Editor you need to have the Adobe Air Runtime installed on your system (Windows XP SP2 or higher / Mac OSX 10.4.11 or higher).

You can get it at this url: <http://get.adobe.com/air/>

To install the Editor simply double click on the **OpenSpaceEditor.air** file and follow the instructions. The application is not yet signed, so a warning will be displayed.

The OpenSpace scene

The following image shows the main elements of the OpenSpace scene:



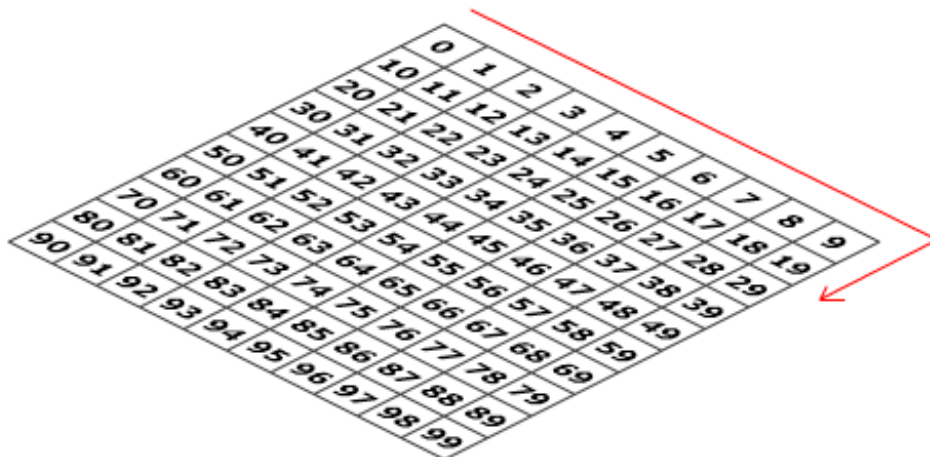
where:

1. Viewport
2. Isometric grid
3. Background layer
4. Foreground layer
5. Empty tile (no graphics)
6. Avatar
7. Tile with associated graphics (a barrel)
8. Tile with oversized graphics (an house)
9. Tile used as an hotspot
10. Scrolling sensor boundaries

The map elements

A virtual world, especially if targeted at the Flash technology, is usually divided into “locations”, where each location is described by its own **map**. OpenSpace handles each map as a separate xml file which is loaded and parsed whenever the *OpenSpace.loadMap* method is called.

Each map is made of **tiles** (which can also be grouped into **supertiles**), usually a **background** and optionally a **foreground**. When the map is parsed and rendered on the stage, tiles are added following the sequence shown in the picture below. The index number indicates the tile depth: the lower the index is, the lower the tile depth is. So, a lower index means that the tile (and everything it contains) is “behind” the tiles with an higher index. Of course this doesn’t respect the nature of a true 3D environment (where, for example, tile 10 would be behind tile 9), but it works pretty well for the purposes of OpenSpace.



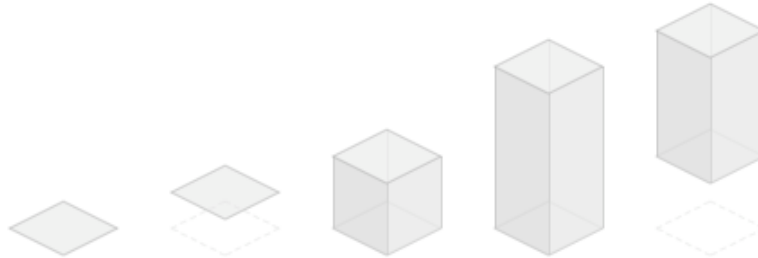
The tiles’ indexes structure is very important and it should always be taken into account when designing a new environment and the objects it will contain, to avoid wrong behavior of map objects (for example wrong overlapping of objects that in a real 3D space would overlap correctly, or wrong overlapping of avatars when moving on the map).

We will talk more about this subject when describing how to handle large, tile-exceeding graphics in the 3D space, after describing tiles and their **skins**.

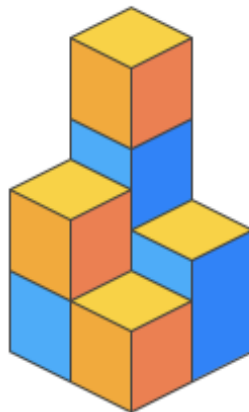
Tiles

In OpenSpace, the tile is a sort of building block (just like the Lego!) which concurs to create the map “architecture”. Each tile has a size, an elevation (the tile height in a 3D perspective), a “viewing angle” and other properties which can be customized. Of course size and viewing angle are common to all the tiles (in fact they are set in the OpenSpace configuration file), while the elevation and the other properties can differ from tile to tile.

Thinking the tile as a building block, we can set a different elevation for both its upper side and its lower side. This leads to countless possibilities, just like in the following image:



Tiles can be stacked one above the other to create complex structures with overpasses or underpasses, combining plain tiles with tiles having a bottom elevation greater than zero. In the following image, orange tiles are stacked upon blue tiles:



Stacking more than two or three tiles is generally not recommended. Tiles composition should also be planned very carefully, to avoid unnecessary tiles to be rendered on the screen, causing performance issues. For example, if you want to place a non-walkable tile representing a tree on an existing walkable tile, instead of simply adding the new tile to the existing one it would be better to remove the previous one and substitute it with the new one.

Another important aspect in building maps is to carefully balance the size of tiles, the size of the map and the size of the viewport: having small tiles, a big map and a wide viewport can cause performance issues, as the number of tiles (together with their skins) on the stage at the same time could be huge.

Skins

A skin is a graphical element contained inside a tile. A skin can be a piece of floor, a tree, a wall, some grass, a greek column, etc. Everything you want to display on the map is a skin (with the exception of the background and the avatar). Most important, a skin can be an entire building too, with boundaries that exceed the tile size (more on this later).

A tile can have no skin, one skin or even more than one skin. Empty tiles (tiles without at least one skin) can be used in combination with the background graphics to simply indicate on which areas of the map avatars can walk, and on which ones they can't.

If more than one skin is assigned to a tile, they will be stacked one above the other. In this way it is possible to create different tiles by combining different skins, just like in the following image:

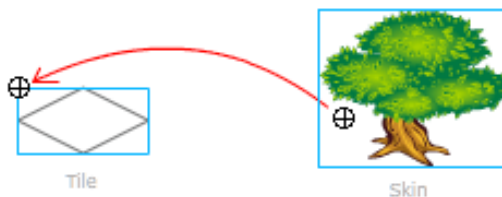


The skins are movieclips or sprites contained in the Flash library of an external swf file (let's call it "skins library file") loaded by the OpenSpace Editor (at author-time) and by the OpenSpace Engine (at runtime); they can be also custom classes which extend those base classes. Each skin must have a unique Class name set in its "Linkage properties", while the Base Class must be *flash.display.MovieClip* or *flash.display.Sprite*. Using sprites instead of movieclips is highly recommended for performance and memory usage reasons. In a tile, the skin's origin is aligned to the (0,0) coordinates of the upper side of the tile; the tile's (0,0) coordinate is shown in the following image:

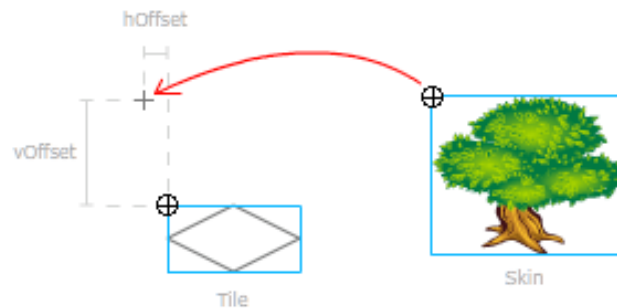


To set the proper skin alignment inside a tile, the best ways are:

1. in Flash, give the graphics contained in the skin sprite a predefined offset with respect to the sprite's origin, so that the skin will have the right positioning automatically;



2. align the graphics in the skin to the sprite's origin and drag the skin in the Skin Editor preview to adjust its position with respect to the tile (alternatively, for a more precise control, you can set the *hOffset* and *vOffset* attributes in the skin properties pane).



The first approach is probably the best, especially when combined with the “auto-generate skins” feature of the OpenSpace Editor (see project setup), because you don’t need to adjust the offset of each skin inside the Editor; also, the artist who creates the skin inside Flash can use a mock-up tile on a guide layer inside the skin sprite to set the correct position of the element (and maybe of other elements around it) with respect to the tile.

As a convention, the skin classes can have some optional methods defined in their timeline's first frame (or inside the class itself in case the skin is linked to a custom class which extends `sprite` or `movieclip`). If available, the OpenSpace Engine can make use of these methods to offer additional customization features at runtime. The conventional methods are:

- **setCustomParams(params:Array):void**

Check the `customParams` skin property in the Tile Editor description.

- **onParentTileSelected(selected:Boolean):void**

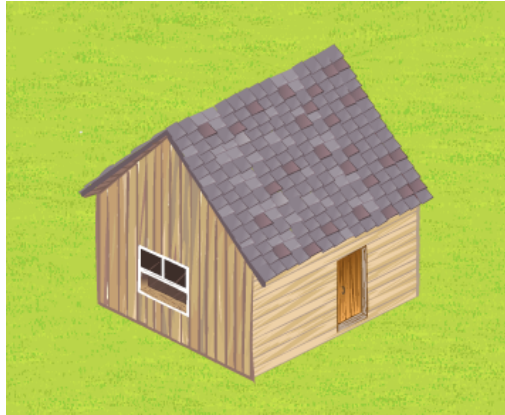
If the skin contains this method, whenever its parent tile is selected in the OpenSpace Engine during runtime (by means of the mouse click or through a call to the `OpenSpace.moveMyAvatar` method), `true` is passed to the method, while `false` is passed to the previously selected tile. This behavior can be useful to highlight the selected tile.

Placing large 2D elements in the 3D space

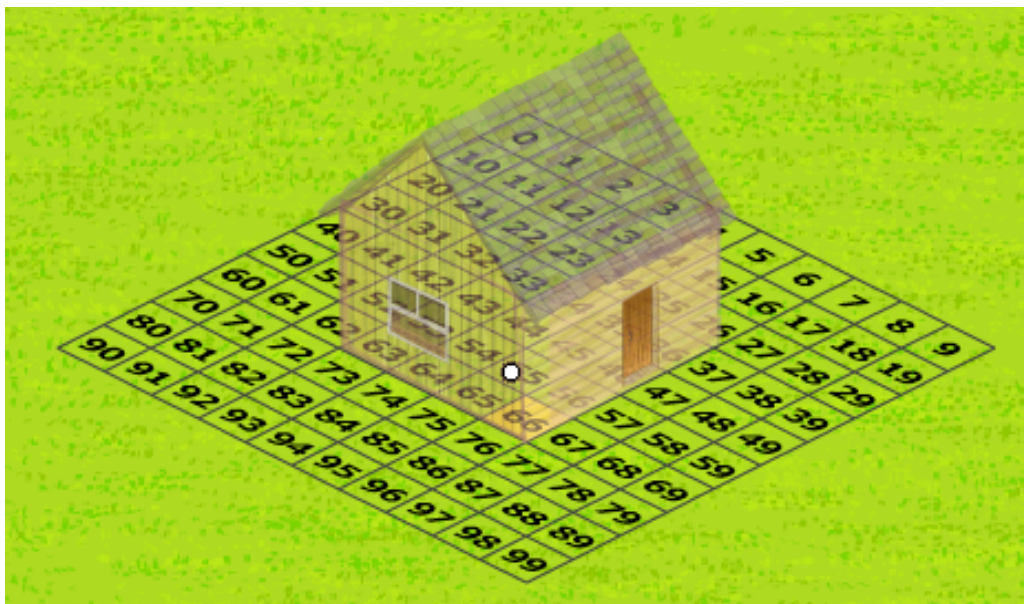
What if we want to create a large 2D object, for example a building, and show it on the map? The task of creating elements that exceed the tile size is not trivial to carry out, because it largely depends on the object size (small width but large depth? or vice-versa?), its shape (convex or concave?), its placement on the map (along the map sides? near other objects?), how the avatars will interact with it (just move along the perimeter? or a more complex interaction?), etc.

This paragraph will describe a couple of ways to approach this task, discussing pro and cons and giving some general guidelines. First of all we have to explain why this task is not just as simple as adding a skin to a tile.

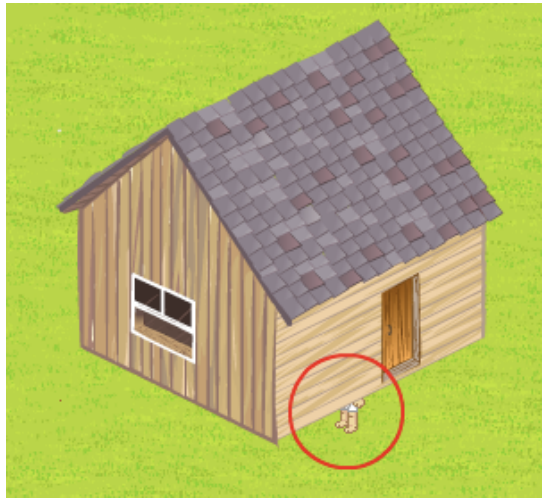
The following image shows a simple country house drawn in Flash as vectors, and then placed on a background.



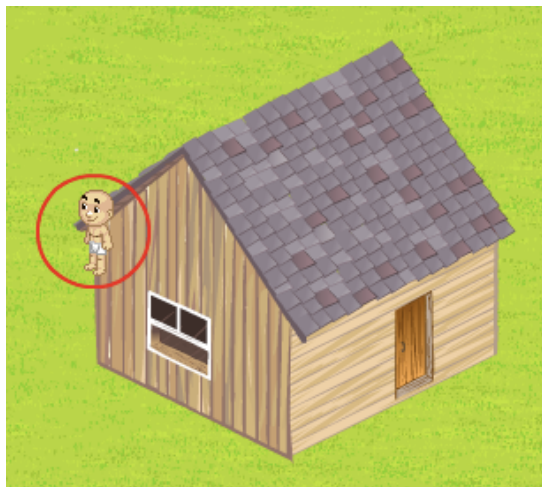
We place it on the isometric grid discussed at the beginning of the chapter, adjusting its alpha value to show the grid behind. In order to simulate what happens in OpenSpace, the house is used as a skin of the yellow highlighted tile.



If we imagine an avatar walking around the house, due to the way avatars are handled (they are contained inside the tile they are on at each instant of their animation), we will immediately see that when the avatar enter tiles 57, 47 and 37 it will be “behind” the house skin (which is part of tile 66), not in front of it as it should be! This is what we would get (supposed avatar is on tile 57):



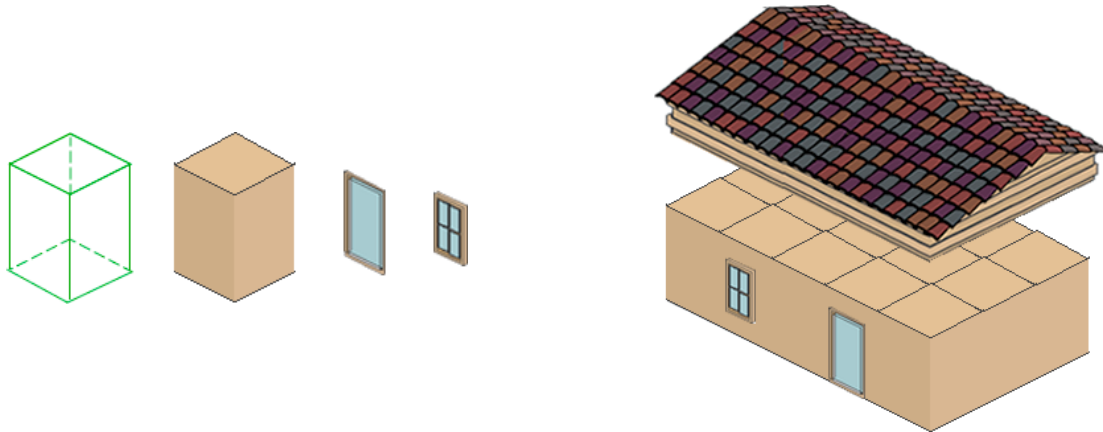
If we attach the skin to tile 63 we have the same result. Instead, if we use the other two corners of the house base (corresponding to tiles 33 and 36), the avatar will overlap in the wrong way when moving behind the house on tiles > 36 and < 63 , for example on tile 51:



Now let's see how to solve this problem.

Tile-based approach

The best way to have large 2D elements on the map is... not to have them! This means that we should divide the skin among all the tiles that constitute the object base (or better, the frontal tiles). This can be done in several ways: the worst is using vertical slices of the bitmap (not recommended!); the best is to create a sort of building blocks to compose each object, taking advantage of the tile's stacked skins feature. The following image is a rough example of the latest technique:



As the picture shows, a single skin is used as basic wall for the house, then additional skins are added to the relevant tiles to create windows, doors and so on. Adding details is the key to make the buildings differ one from the other and make their appearance less “tiled” (for example we could add cracks on the walls, some ivy, a balcony, have different shapes, etc.). We could even add a second floor to some buildings, by stacking tiles one above the other.

Moreover, in the above example the roof is a single, non-tiled bitmap, which can be used in a tile above the front corner. This is possible because avatars height will never make them “disappear” under the roof (oppositely to what happens in the first of the images showing the red circle).

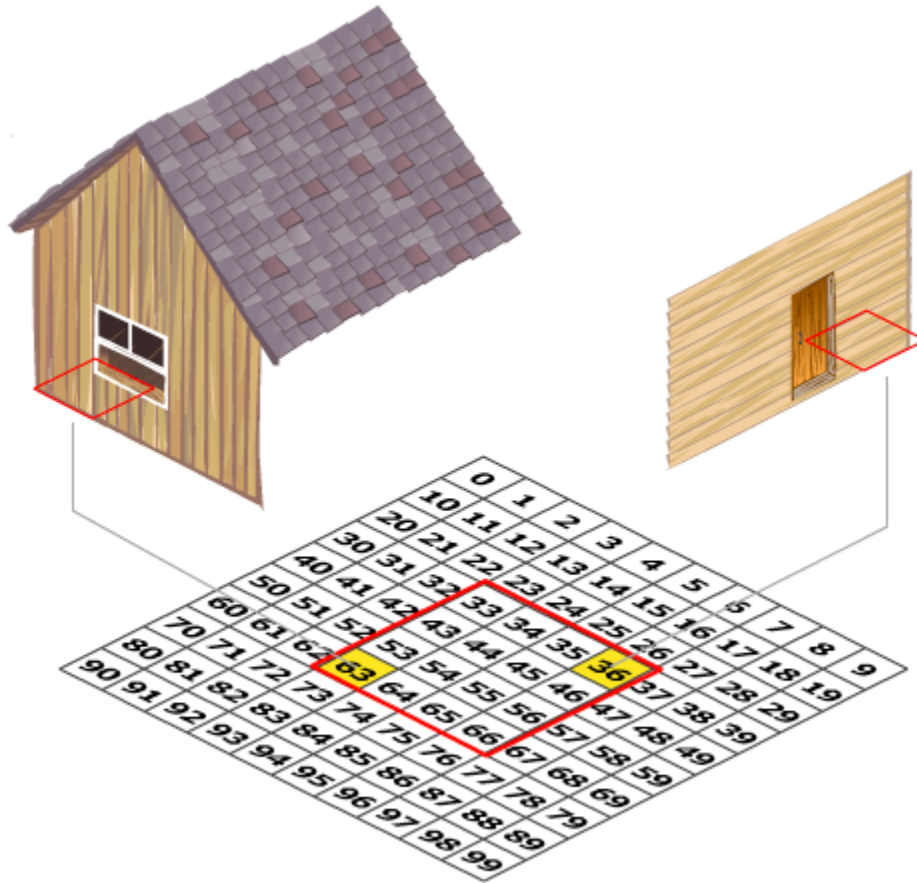
This technique has the big advantage of allowing a great reusability of each element, letting you to build different objects from a reduced number of skins, thus reducing the size of the tiles library and the memory consumption. Also, this technique allows the creation of complex elements, like overpasses, bridges and underpasses.

On the other hand, all this requires graphic designers to make an effort to create the building blocks properly.

Skin separation approach

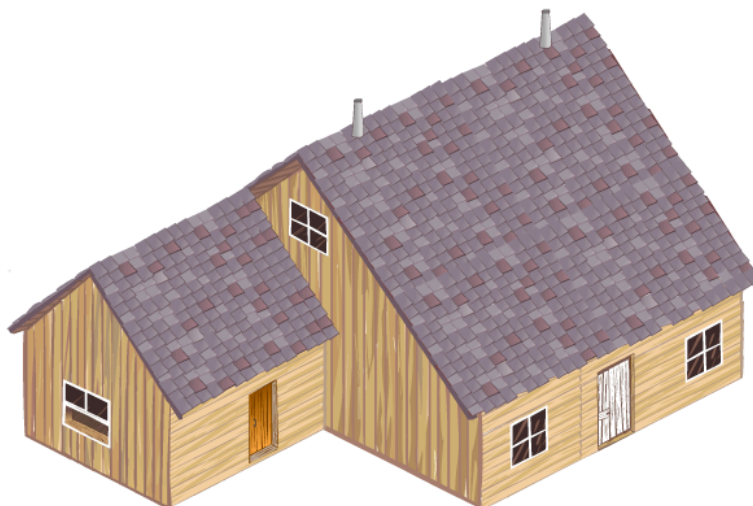
In order to keep a better control over the objects' graphics, avoiding the common defects of a tiled approach (repetitiveness, above all) you can adopt this second approach, which consists in dividing the skin into two parts, following the main isometric directions. How to divide the skin largely depends on its shape: the only suggestion is to reason on the numbered isometric grid, to understand how to make the different parts correctly overlap with surrounding objects or avatars.

Here is an example, which makes use of the previous country house:



The house has been divided into two skins, assigned to different tiles (63 and 36). This solves the overlapping problems seen previously.

This approach may not work properly in some cases, for example if the building's structure contains a concavity, like in the following image:



In this case, dividing the skin between two tiles is not enough (try to reason on the numeric grid to see why), but you can still divide it into four parts, as if we had two distinct buildings (both convex) on which to use the described technique:

**NOTE**

Using the “skin separation” approach with bitmap graphics containing transparencies (usually the bitmap’s bounding box is filled with transparent pixels), you may experience this issue: the transparent part of the bitmap catches the mouse click, so you can’t make your avatar move on the tiles covered by the bounding box of that skin.

In order to solve this problem, simply remove the background of the bitmaps after importing them into Flash, following this tech note from Adobe:

http://www.adobe.com/go/tn_12804

Whatever of the proposed techniques is used, a good approach is to “compose” the large objects by grouping the tiles it consists of (tiles 63 and 36 in the above example), so that they can be easily instantiated more than once on one or more maps. This can be achieved using the Supertiles Editor, as described in the next chapter.

Background and foreground

OpenSpace uses separate background and foreground layers: background is placed below all the tiles, while foreground is always above.

A separate layer for background let the map designers have a better control over the environment appearance, without being subject to the restrictions and defects of a tiled approach. Taking this to the extreme, an environment could be made of just a background covered with empty tiles (tiles without skin) which simply set where an avatar can or

cannot walk. Also, having a separate background allows you to fill with continuous graphics (which scroll together with the map) those parts of the scene which aren't covered with tiles (typically the viewport corners, due to the fact that the isometric map has a diamond shape).

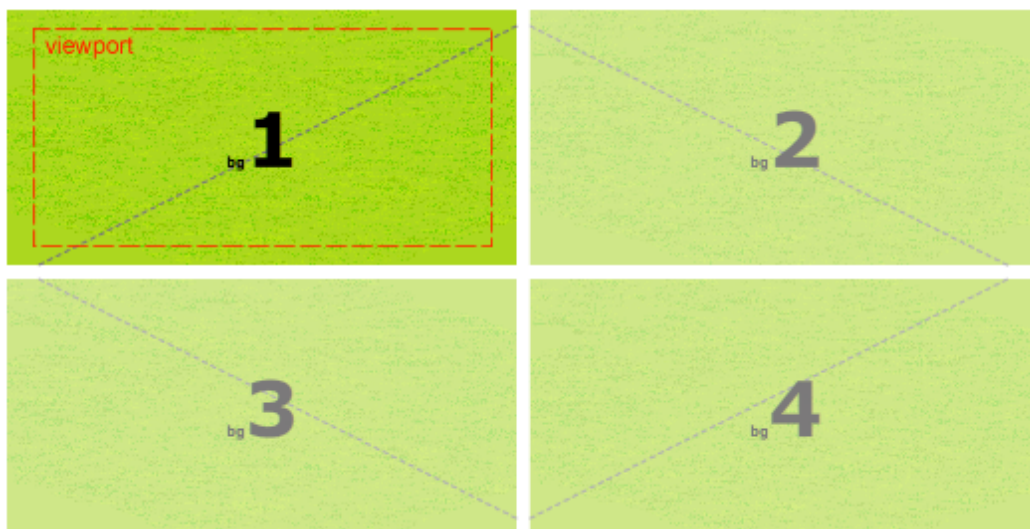
As regards foreground, if its overall size is smaller than the background overall size, a parallax effect is applied during the map scrolling.

Both background and foreground graphics should be split into several parts in order to improve the scrolling performance. In fact, whenever one of the background/foreground parts is out of sight, the OpenSpace Engine removes it from the display list, therefore avoiding to render it during the scrolling.

Background and foreground parts are placed on an orthogonal grid (not isometric!) in the Background Editor, thus each one of them must have the same size (width and height).

The smaller the size is (with respect to the viewport size), the better it is.

The following image shows an example of background divided into four parts:



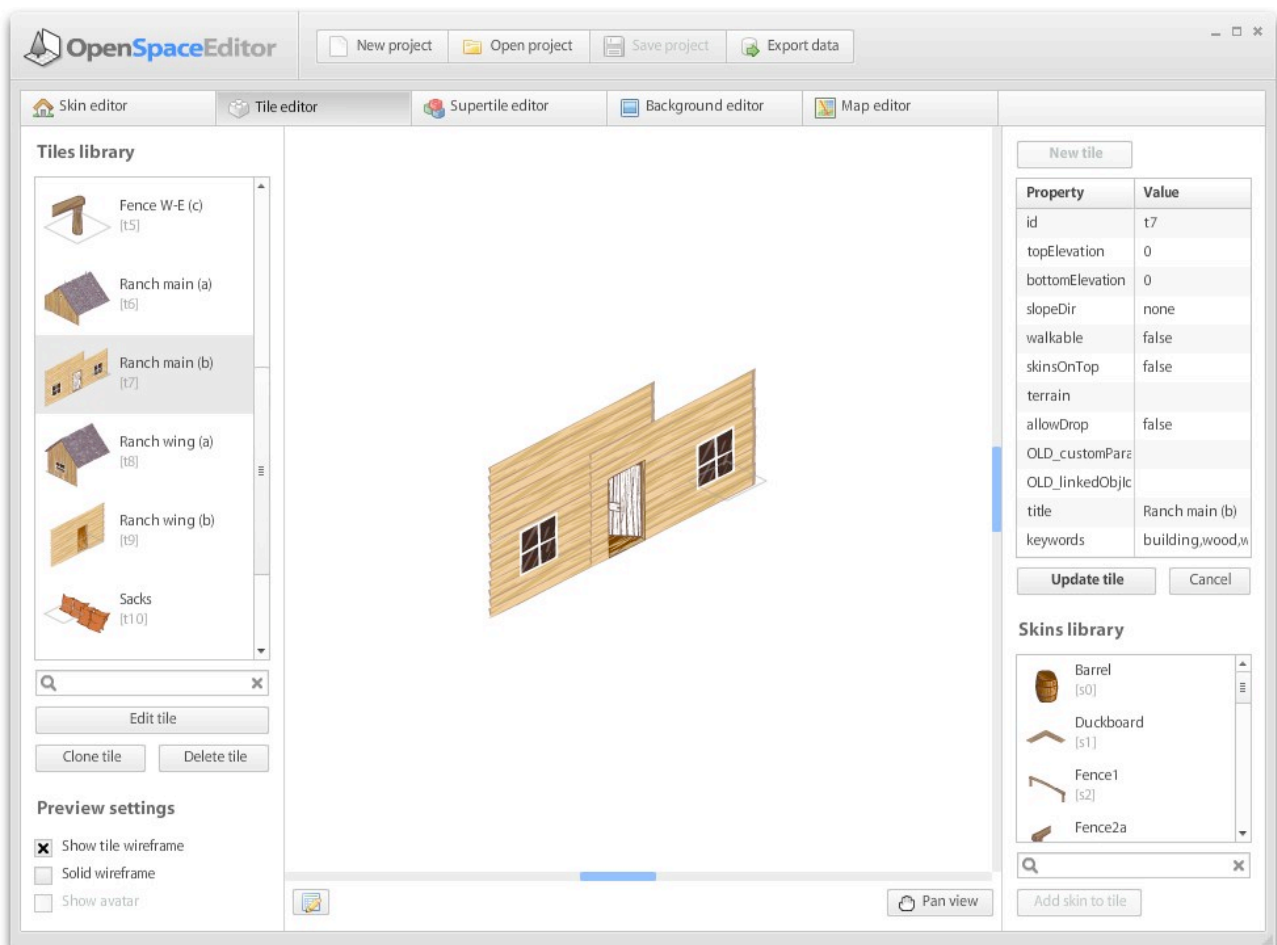
The background / foreground parts are movieclips or sprites contained in the Flash library of an external swf file (let's call it "backgrounds library file") loaded by the OpenSpace Editor (at author-time) and by the OpenSpace Engine (at runtime). Each part must have a unique Class name set in its "Linkage properties", while the Base Class must be *flash.display.MovieClip* or *flash.display.Sprite*. Using sprites instead of movieclips is highly recommended for performance and memory usage reasons.

At author-time, the OpenSpace Editor processes the backgrounds library swf to retrieve all the available parts automatically.

Using the Editor

The OpenSpace Editor is made of a project setup panel, a number of sub-editors (Skin, Tile, Supertile, Background/Foreground and Map Editors) and an export panel.

All the sub-editors have a similar layout, divided into three columns as shown in the following image:



The left column contains the library of items created with the selected Editor (in the above example the Tile Editor) and specific controls to modify the preview settings. The items' list features a text input at the bottom which allows the list filtering (the entered words are checked against the item's id, title and keywords - see the properties pane description); the "Edit", "Clone" (where available) and "Delete" buttons can be used to edit, duplicate or remove an existing item respectively (please notice that in all cases the "Save project" button must be pressed to commit the changed before closing the OpenSpace Editor).

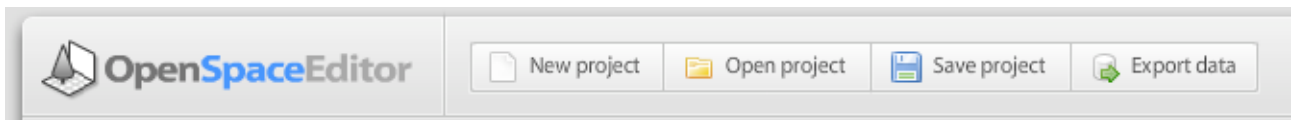
The center column shows the preview of the currently edited item, that the user can interact with. The column's footer contains a button to enable preview area panning and additional informations where needed. The Tile Editor and Map Editor footers also contain

a button (on the left) to show the “Instance properties” panel, This panel is used to set additional properties for the instances of skins, tiles and groups that have been dropped into the currently edited item, for example a tile on a map (see the description of each Editor for further details).

The right column contains a properties pane for the currently edited item and, below it, the library of items from the other editors that can be added to the current item, for example the library of tiles that can be added to a map.

The usage of the Editor is quite straightforward and self-explanatory; the following paragraphs describe each section.

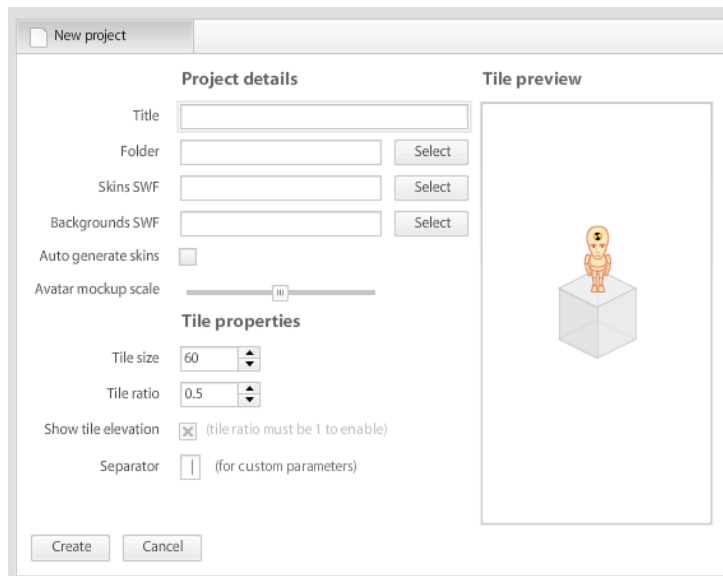
Main menu



The main menu contains the button for the basic Editor actions: create a new project, open an existing project, save the current project and export the project’s data to be used in the OpenSpace Engine.

New project

When creating a new project, the form shown in the image below must be filled in. When pressing the “Create” button, a file with extension **.osp** is saved in the selected folder (the name of the file is generated from the project title by stripping invalid characters). After the project is created it is not possible to change those values visually. In case you need to make changes, edit the **.osp** file with a text editor.



The project details are:

Title	The project name.
Folder	The folder where the project file will be saved. The project file (extension: .osp) is an xml file containing the project details configuration data and libraries data (skins definition, tiles definition, etc.).
Skins SWF	The path to the swf file containing the graphical elements (sprites or movieclips) that will be used to create the tiles' skins. Each sprite/movieclip must have a Class name entered in the "Linkage properties" so that the Skin Editor can access it.
Backgrounds SWF	The path to the swf file containing the graphical elements (sprites or movieclips) that will be used to create the maps' backgrounds and foregrounds. Each sprite/movieclip must have a Class name entered in the "Linkage properties" so that the Background Editor can access it.
Auto generate skins	When the project is created and this flag is set, the Skin Editor will automatically generate all the skins by retrieving all the sprites / movieclips which have a Class name set in their "Linkage properties" in the provided swf file. This is a one-time process: all the skins corresponding to the graphics added to the swf file at a later time must be created manually inside the Skin Editor.
Avatar mockup scale	Set the scaling of the mockup avatar with respect to the tile size. <i>NOTE: this feature is not currently used in the OpenSpace Editor</i>
Tile size	The width, in pixels, of all the tiles for the current project.
Tile ratio	The aspect ratio of all the tiles for the current project. This value can be lesser than or equal to 1. If equal to 1, the tile's height is the same as its width (the tile looks like a square rotated of 45 degrees).
Show tile elevation	If this flag is set, the tile elevation is always visible even if the aspect ratio is 1. Deselect the checkbox for a top-down view.
Separator	The character used to separate the parameters set in the skin's <i>customParams</i> property.

Open project

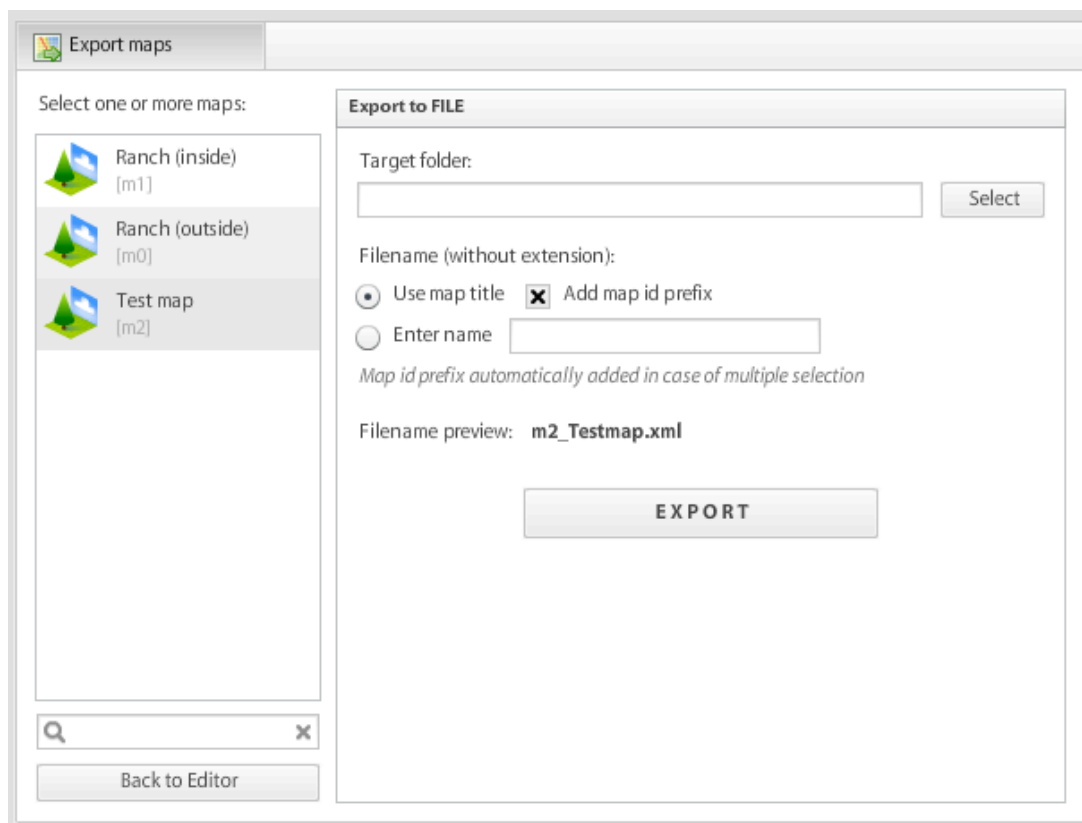
Click the “Open project” button and select a project file to open it. Please notice that at startup the OpenSpace Editor registers itself as default application for the **osp** file type, so after the first execution it is possible to open .osp files by simply double clicking them.

Save project

Whenever a change occurs in one of the Editors, the save button is enabled. Clicking it causes all the libraries of the various Editors to be saved in the project’s file.

Export data

The export panel allows the creation of the map xml file to be loaded in the OpenSpace Engine by means of the *OpenSpace.loadMap* method. Multiple maps can be exported at once, in separate files.



Skin Editor

The Skin Editor allows the creation of the skins to be used in the Tile Editor. Just like in the following editors, when clicking on the “New” button the properties pane is filled with the default properties (for the skin, in this case) and an empty preview is created. When clicking the “Create” or “Update” buttons the skin is added to the library on the left.

Skin properties

The following table describes the skin properties:

id	The skin's unique id, automatically generated by the Editor.
className	The Class name entered in the "Linkage properties" of the skin sprite / movieclip during creation in the Flash authoring tool; the class can be selected in the dropdown or, in case it wasn't retrieved automatically by the OpenSpace Editor, it can be entered manually.
cache	<p>If set to <i>true</i>, the <code>cacheAsBitmap</code> property of the skin is set to <i>true</i>; default is <i>false</i>.</p> <p>IMPORTANT: if the skin contains vector graphics, this attribute should always be set to <i>true</i>, unless the graphics are animated continuously (but this is highly discouraged, due to possible performance issues); if the skin contains bitmap graphics, this attribute should be set to <i>false</i> to avoid unnecessary memory usage. In case of a mixed situation, for example a skin containing a bitmap house and a vector animated door, as the door animation is not continuous (usually it opens when the avatar is in front of it, or when the user clicks it) and its dimensions are small with respect to the house bitmap, the cache attribute should be set to <i>true</i>.</p>
hOffset / vOffset	<p>The skin's horizontal and vertical offsets with respect to the (0,0) coordinates of the tile. In case the skin is added to a tile with top elevation greater than 0, the (0,0) coordinates of the top side are taken into account for skin alignment.</p> <p>These properties can be set by changing the values in the properties pane or by dragging the skin's preview.</p>
title	The skin's name that will be displayed in the skins library.
keywords	The keywords that describe the skin, useful to filter the skins library. All the keywords must be separated by a comma.

Other features

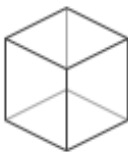




Using the "Show tile mockup" checkbox on the left it is possible to show / hide a reference tile that can be useful to properly set the position that the skin will have once added to a tile using the Tile Editor.

Tile Editor

The Tile Editor allows the creation of the tiles available in the Supertile and Map Editors.

Tile properties

The following table describes the tile properties:

id	The tile's unique id, automatically generated by the Editor.
topElevation	The elevation of the upper side of the tile; it must be greater than or equal to <i>bottomElevation</i> .
bottomElevation	The elevation of the lower side of the tile; it must be greater than or equal to 0. If the <i>bottomElevation</i> is greater than 0, underpasses can be created (if the avatars' movieclip height is lower than the <i>bottomElevation</i> they will be able to walk below the tile).
slopeDir	<p>The tile's slope direction. If the tile is a slope, it goes from <i>bottomElevation</i> to <i>topElevation</i>. The available values are:</p> <div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;"> <p>-1 or missing (not a slope)</p>  </div> <div style="text-align: center;"> <p>0 (north)</p>  </div> <div style="text-align: center;"> <p>2 (east)</p>  </div> <div style="text-align: center;"> <p>4 (south)</p>  </div> <div style="text-align: center;"> <p>6 (west)</p>  </div> </div> <p>By default a tile is not a slope.</p>
walkable	If set to <i>true</i> , the avatars will be able to step on the tile. The default value is <i>false</i> .
skinsOnTop	If set to <i>true</i> , when an avatar enters the tile, it will be placed behind the tile's skins (by default the avatar is always in front of the skins of the tile it is currently above). The default value is <i>false</i> .
terrain	<p>Assign a terrain type to the tile; this affects the pathfinding and avatars speed according to the OpenSpace Engine configuration.</p> <p><i>NOTE: this feature is not currently supported by the OpenSpace Engine</i></p>
allowDrop	<p>If set to <i>true</i>, during runtime map editing the user is allowed to stack a tile above the current one. This parameter can be useful to avoid map inconsistencies like, for example, a chair placed above a tree.</p> <p>The default value is <i>true</i>.</p> <p><i>NOTE: this feature is not currently supported by the OpenSpace Engine</i></p>

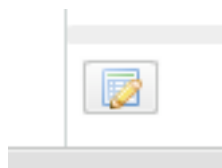
OLD_customParams	<p>A string containing custom tile parameters. The parameters must be divided by the separator declared in the OpenSpace configuration file. Whenever an OpenSpace event contains the reference to a tile (for example the avatar movement related events, see the OpenSpace API), an array containing these parameters can be retrieved using the Tile.customParams property.</p> <p>These parameters can be useful to identify specific tiles (like hot spots) on the map in order to take some action. For example a certain tile could cause a new map to be loaded when the avatar steps on it, or a door inside a skin to display an opening animation, or a sound to be played, etc.</p> <p><i>NOTE: this parameter has been temporary added for compatibility with the current OpenSpace Engine, and will be removed in the next versions due to the adoption of a different approach</i></p>
OLD_linkedObjId	<p>This attribute can be used to create a logical connection between the tile and a skin, even if belonging to a different tile. This is useful to create hot spots (or “sensor tiles”) which cause an animation inside a skin to be played (for example a door opening).</p> <p>See the skin instance properties for more details.</p> <p><i>NOTE: this parameter has been temporary added for compatibility with the current OpenSpace Engine, and will be removed in the next versions due to the adoption of a different approach</i></p>
title	The tile’s name that will be displayed in the tiles library.
keywords	The keywords that describe the tile, useful to filter the tiles library. All the keywords must be separated by a comma.

Handling skin instances

A skin instance from the skins library on the right can be added to the currently edited tile by double clicking it or selecting it and then pressing the “Add skin to tile” button.

To remove the skin instance, click on it in the preview to make the selection highlighter appear and then click the X button in the corner or press the CANC key.

When a skin instance is selected in the preview, it’s possible to edit the instance properties by clicking the button shown in the following image to make the instance properties panel appear:



The editable instance properties are:

Name (objectId)	<p>A unique name to be assigned to the skin. This name is useful to access the skin's sprite / movieclip from the main application, for example to show an animation when a certain event occurs. For example a skin showing a building could contain a door that should open when an avatar steps on a certain tile: using the <i>OLD_linkedObjId</i> attribute (check the tile properties) on the tile, a connection between the tile and the skin can be set, so that at runtime, when the avatar stops on the tile, the reference to the skin can be retrieved and the door animation played.</p> <p>IMPORTANT: this property must be unique; two skins on the same map can't have the same name.</p>
Custom params	<p>A string containing custom skin parameters. The parameters must be divided by the separator declared in the project's details. In order to handle the custom parameters, the skin class must contain a <i>setCustomParams</i> method which takes an array as argument.</p> <p>When the skin instance is created by the OpenSpace Editor or Engine, if the <i>setCustomParams</i> method exists, it is called and the custom parameters are passed in form of an array.</p> <p>These parameters can be useful to setup skins at runtime: for example the skins library could contain a "Chair" class and a custom parameter could be used to set its color (instead of having a different "Chair" class for each possible color).</p>
Triggers	<i>NOTE: this feature is not currently supported by the OpenSpace Engine</i>

Other features

Using the "Show tile wireframe" and "Solid wireframe" checkboxes on the left it is possible to show / hide the tile's frame.

The selection highlighter displays the skin's library id (where "s" stands for "skin") and title.

Supertile Editor

The Supertile Editor allows the creation of the supertiles available in the Map Editor.

A supertile is a sort of mini-map that allows tiles grouping for faster editing when a complex object is made of two or more tiles (see the previously described techniques), and it must be used more than one time on the same map (an house, a section of a fence, a park bench, a thicket, etc.).

Supertile properties

The following table describes the supertile properties:

id	The supertile's unique id, automatically generated by the Editor.
width / height	The supertile' size, expressed in number of tiles.
title	The supertile's name that will be displayed in the supertiles library.
keywords	The keywords that describe the supertile, useful to filter the supertiles library. All the keywords must be separated by a comma.

Handling tile instances

A tile instance from the tiles library on the right can be added to the currently edited supertile by selecting the destination position (an empty cell on the grid or a previously added tile) and:

- double clicking the tile in the library; or
- selecting the tile in the library and then pressing the “Add tile to supertile” button.

It is also possible to add a tile without previously selecting the destination:

- drag the tile from the library and drop it on the destination position (*drag&drop* can be used to move tiles already on the map too); or
- select the tile in the library and then click on the destination while keeping the SHIFT key pressed.

To remove the tile instance:

- click on it in the preview to make the selection highlighter appear and then click the X button in the corner or press the CANC key; or
- click on the tile while keeping the ALT key pressed.



Insertions optimization

The OpenSpace Editor automatically optimizes tiles insertions. Check the Map Editor description for additional informations.

Other features

The “Show base grid” checkbox on the left can be used to show / hide the supertile's grid, while using the “Show tile wireframe” and “Solid wireframe” checkboxes its possible to show / hide the frame of all the tiles on the map. When the solid wireframe is displayed, walkable tiles are highlighted in green (top face only).

When the mouse rolls over a tile, its coordinates are displayed in the center column's footer.

The selection highlighter displays the tile's library id (where “t” stands for “tile”) and title and an icon in the bottom right corner which indicates if the tile is walkable  or not .

Pressing the ESC key deselects the currently selected tile.

Using the “Fill base layer” in the bottom right corner of the Editor it's possible to fill the supertile with the selected tile; please notice that if it is pressed after the supertile editing is already in progress, it causes all the bottom-most tiles to be removed and substituted by the selected one.

Background Editor

The Background Editor allows the creation of both backgrounds and foregrounds to be assigned to a map in the Map Editor.

Background properties

The following table describes the background / foreground properties:

id	The background / foreground's unique id, automatically generated by the Editor.
columns / rows	The size of the background / foreground expressed in number of "parts" that constitute it.
colWidth / rowHeight	The size (width and height respectively) of the parts that constitute the background / foreground.
cache	<p>If set to <i>true</i>, the <code>cacheAsBitmap</code> property of the background parts is set to <i>true</i>; default is <i>false</i>.</p> <p>IMPORTANT: if the background / foreground contains vector graphics, this attribute should always be set to <i>true</i>, unless the graphics are animated continuously (but this is highly discouraged, due to possible performance issues); if the background / foreground contains bitmap graphics, this attribute should be set to <i>false</i> to avoid unnecessary memory usage.</p>
title	The background / foreground's name that will be displayed in the backgrounds library.
keywords	The keywords that describe the background / foreground, useful to filter the backgrounds library. All the keywords must be separated by a comma.

Handling background parts

The background / foreground parts from the parts library on the right can be added to the currently edited background by selecting the destination cell and:

- double clicking the part's name in the library; or
- selecting the part in the library and then pressing the "Add part to background" button.

To remove a part, click on it in the preview to make the selection highlighter appear and then click the X button in the corner or press the CANC key.

Other features

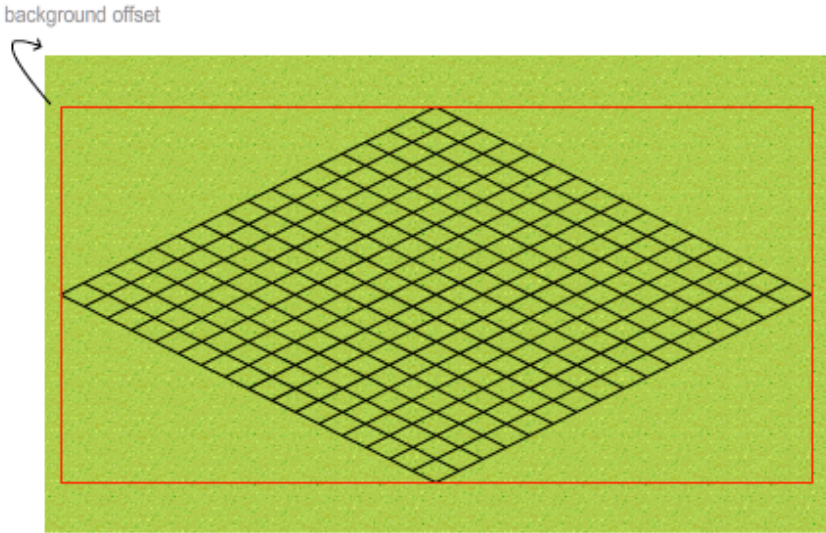
The "Show grid" checkbox on the left can be used to show / hide the background's grid.

Map Editor

The Map Editor allows the creation of the maps library.

Map properties

The following table describes the map properties:

id	The map's unique id, automatically generated by the Editor.
width / height	The size of the map, expressed in number of tiles.
backgroundId	The id of the background, selectable among those available in the backgrounds / foregrounds library.
bgHOffset / bgVOffset	<p>The background's horizontal and vertical offsets with respect to the (0,0) coordinates of the map. The map's (0,0) coordinates is located in the top, left corner of the map's bounding box, as shown in the following image:</p> 
foregroundId	The id of the foreground, selectable among those available in the backgrounds / foregrounds library.
title	The map's name that will be displayed in the maps library.
keywords	The keywords that describe the map, useful to filter the maps library. All the keywords must be separated by a comma.

Handling tile instances

A tile instance from the tiles library on the right can be added to the currently edited map by selecting the destination position (an empty cell on the grid or a previously added tile) and:

- double clicking the tile in the library; or
- selecting the tile in the library and then pressing the “Add to map” button.

It is also possible to add a tile without previously selecting the destination:

- drag the tile from the library and drop it on the destination position (*drag&drop* can be used to move tiles already on the map too); or
- select the tile in the library and then click on the destination while keeping the SHIFT key pressed.

To remove the tile instance:

- click on it in the preview to make the selection highlighter appear and then click the X button in the corner or press the CANC key; or
- click on the tile while keeping the ALT key pressed.

When a tile instance is selected in the preview, it's possible to edit the following properties in the instance properties panel:

Name	<i>NOTE: this feature is not currently supported by the OpenSpace Engine</i>
Allow remove	<p>If selected, during runtime map editing the user is allowed to remove the tile from the map. This parameter can be useful to create fixed map items that can't be removed by users at runtime. If the tile is part of a group (see later), this property can't be changed and must set globally for the group.</p> <p><i>NOTE: this feature is not currently supported by the OpenSpace Engine</i></p>
Access point	<p>If selected, the tile is registered among the access points for the map. When the user's avatar is created at runtime, after the map rendering, if coordinates are not passed, the OpenSpace Engine randomly selects one of the available access point and places the avatar on the corresponding tile.</p> <p>IMPORTANT: each map must have at least one access point defined.</p>
Triggers	<i>NOTE: this feature is not currently supported by the OpenSpace Engine</i>

Handling supertile instances

A supertile instance from the supertile library on the right can be added as a “group of tiles” (see box below) to the currently edited map by selecting the destination position (an empty cell on the grid or a previously added tile) and:

- double clicking the supertile in the library; or
- selecting the supertile in the library and then pressing the “Add to map” button.

It is also possible to add a supertile without previously selecting the destination:

- drag the supertile from the library and drop it on the destination position (*drag&drop* is not available for groups already on the map); or

- select the supertile in the library and then click on the destination while keeping the SHIFT key pressed.

IMPORTANT

When a supertile is added to a map, it is converted into a **group** of tiles, losing any reference to the original supertile. This means that if the supertile is later updated (changing size, adding/removing tiles, etc.) in the Supertile Editor, the changes won't be reflected on the maps where that supertile was added.

ADDITIONAL NOTES

In order to change the instance properties of tiles belonging to a group (as described in the *Handling tile instances* paragraph), keep the CONTROL key (in Windows) or the COMMAND key (in Mac OSX) pressed while clicking on the tile. In this way a single tile can be selected instead of the entire group.

If a tile belonging to a group is dragged in a different position on the map, it still belongs to the group which is re-shaped accordingly.

If a tile belonging to a group is removed from the map, that tile is removed from the group too.

To remove a group of tiles click on one of its composing tiles in the preview to make the selection highlighter appear and then click the X button in the corner or press the CANC key.

When a group of tiles is selected in the preview, it's possible to edit the following properties in the instance properties panel:

Allow remove	<p>If selected, during runtime map editing the user is allowed to remove the group from the map.</p> <p>This parameter can be useful to create fixed map items that can be removed by users at runtime.</p> <p>Setting this property will set the corresponding property of all the tiles belonging to the group.</p> <p><i>NOTE: this feature is not currently supported by the OpenSpace Engine</i></p>
---------------------	---

Insertions optimization

The OpenSpace Editor automatically optimizes tiles / supertiles insertions. Under certain circumstances, when a tile is added to the map and the destination coordinates already contain a tile, the existing tile may be substituted, instead of the new tile being stacked upon it. This allows the reduction of the number of tiles on the map by removing the unnecessary ones, thus improving performance.

Optimization occurs in the following cases:

- a plain (topElevation = 0) tile is inserted above an identical tile (actually in this case the existing tile isn't substituted and the insertion is simply cancelled);
- a tile with bottomElevation = 0 is inserted above a plain, not skinned tile;
- the tile contained in the destination coordinates doesn't belong to a group.



Version 1.0.1



If the added tile is later removed, the previous tile won't be restored.

Other features

The "Show base grid" checkbox on the left can be used to show / hide the map's grid, while using the "Show tile wireframe" and "Solid wireframe" checkboxes it's possible to show / hide the frame of all the tiles on the map. When the solid wireframe is displayed, walkable tiles are highlighted in green (top face only).

The "Show background" and "Show foreground" checkboxes can turn on and off the background and foreground respectively.

When the mouse rolls over a tile, its coordinates are displayed in the center column's footer.

The selection highlighter displays the map item's library id (where "t" stands for "tile" and "g" for group) and title. If the selected item is a tile, an icon is displayed in the bottom right corner indicating if the tile is walkable  or not ; if the selected item is a group, an icon containing a G is displayed in the bottom right corner. Also, if the item is a group, its title is the same of the supertile that originated the group.

Pressing the ESC key deselects the currently selected tile / group.

Using the "Fill base layer" in the bottom right corner of the Editor it's possible to fill the map with the selected tile; please notice that if it is pressed after the map editing is already in progress, it causes all the bottom-most tiles to be removed and substituted by the selected one.