

TweeFly 0.8 BETA Documentation

stonedrum.de, 2018

# Introduction

*TweeFly* is an UI based setup tool for interactive stories written with *Twee2*. The idea behind it is to have a tool that generates all the required Twine, CSS and JavaScript files to handles various tasks like creating an inventory, a shop system, or a daytime cycle.

## Editions

There exist two different editions: Free and Professional. The *Professional Edition* includes features that the *Free Edition* does not (surprise). See the following table for all the feature’s differences:

Table 1: Feature comparison of Free and Professional Edition

|  |  |  |
| --- | --- | --- |
| Feature | Free version | Professional version |
| Inventory | X | X |
| Shops | X | X |
| Money | X | X |
| Clothing |  | X |
| Stats |  | X |
| Daytime |  | X |
| Jobs |  | X |
| Characters |  | X |
| Custom captions |  | X |
| Story configuration | X | X |

Both editions can be purchased and/or downloaded from [stonedrum.de](http://www.stonedrum.de).

# Quick start

*TweeFly* is pretty easy to setup. Install the application and start it. You see a *menu* on the left and some *tabs* on the right. First, check the features you want to use in your game (e.g. *inventory*, *shops*, *money* in the *Free Edition*). Some features depend on others and are selected automatically (if you want to use a *shop system* you need to activate the *money* feature as well. It is selected by *TweeFly* if it has not been checked yet).

Now add your items or shops in the corresponding tabs and select *Display link in sidebar* on the *inventory* tab so that you can access your inventory any time in the game. Once you did that, give your story a name in the edit field in the *menu* on the left and name your main story file that has to include the *::Start* paragraph which is the entrance point to your story.

If you wish to play the story after the generation mark *Run after generation* and click *Generate and build*. *TweeFly* will then ask for a folder to put the files into. This should be the folder containing your main story file. If every file was found and if there are no compilation errors your story will open up containing a link to your inventory on the left.

From now on you can call the build.bat in the folder selected when making changes to your story. You only require to open *TweeFly* again when you want to change items, or shops, or change the behavior of the scripts generated.

# TweeFly in-depth guide

In this section we will go through all functions of *TweeFly* and learn how to use them. There are various combinations you can use so this guide will create a scenario where we use them all. From time to time there are differences between the Professional and the Free Edition. Features only included in the Professional Edition are marked with a -P-.

## The menu

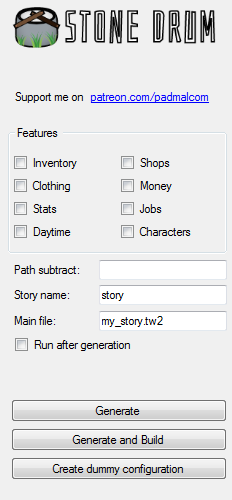


Figure 1: The TweeFly menu

The menu is found on the left of the main window and includes all features that can be selected:

* Inventory
* -P Clothing
* -P Stats
* -P Daytime
* Shops
* Money
* -P Jobs
* -P Characters

As mentioned at the beginning some features require other features. Those are selected automatically by *TweeFly*.

The field **Path subtract** is used when you specify images e.g. to display an item. Most certainly, the URL of an image might differ on the developer's and the player’s computer. As a consequence, using relative paths is the way you should go. When picking an image for an item the dialog adds the entire file path as property. To make this path a relative path, you can leave it as it is and specify a string to subtract from the beginning of the image’s path. An example:

You select an image from:

*C:\Users\developer\mystory\img\chocolate.png*

And specify a subtraction path:

*C:\Users\developer\mystory\*

Then the path referenced in the code generated is:

*img\chocolate.png*

Thus, you can place all your images when packaging your story in a folder *img* and its path is valid whoever plays your story.

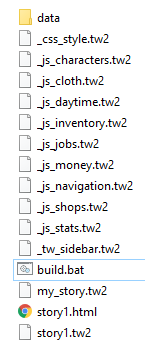
**Story name** specifies the name of the story as it is shown in the sidebar in your game and sets the name of the html file generated.

Figure 2: Folder structure of a TweeFly project

**Main file** is the *Twee2* file (\*.tw2) that contains your story. This main file should contain a paragraph *::Start*. I had to design the program like that to make sure that your story is not overwritten any time you regenerate the *TweeFly* code. So, your main file remains untouched by *TweeFly* but every file around the main file is created once you click *Generate*. If the main file does not exist in the folder where you generate your *TweeFly* code the process generates a valid file for you so that the *Twee2* compiler does not run into errors.

A classic folder structure for a project should look like the image on the left (the folder *data* is added manually and recommended to contain audio and image files).

You can generate your code in several ways. You either click **Generate** to have *TweeFly* only create all the files required for the build. **Generate and build** creates the files and executes *Twee2* to create the html file, too. If you select **Run after generation**, *TweeFly* executes the html file.

[BETA] **Create dummy conf** is a button to fill *TweeFly* with some test values so that you (and I) can quickly test the generation process. It will most certainly disappear from the tool, soon.

## Inventory tab

The *inventory tab* is there to define items and the look of your inventory. Above the items list you find six checkboxes. **Display link in sidebar** tells *TweeFly* if there should be a link to the inventory created in your sidebar so that you can access the inventory any time in your game. **Display in sidebar** displays the entire inventory (which can be very large depending on the number of your items) in the sidebar. Each item has several properties like a name, a category and so on. **Inventory sidebar tooltip** says if these properties are shown when you hover over an item image in the sidebar item list.

Each item has one to three blank **skills** that can contain an arbitrary value for the item so that you can integrate some own logic in your game using those skills. If any skill should be used in your game check the corresponding checkbox. Otherwise they are not rendered in the game later on.

The list **Display in inventory** names all item properties that appear in the item table in your game. Those properties still exist in the item definition but are not shown when the property is not checked.

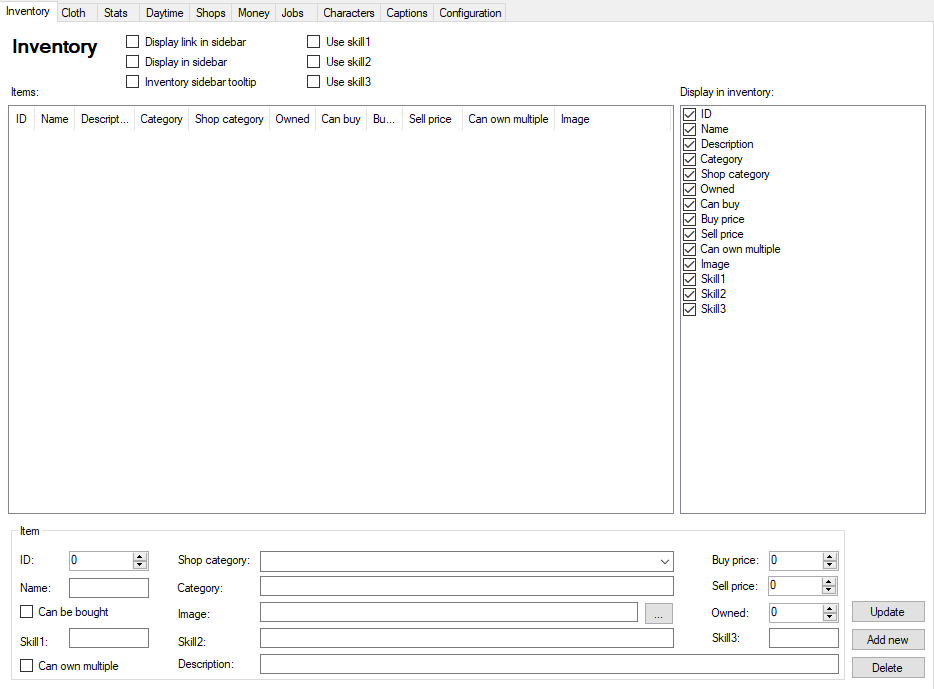


Figure 3:The inventory tab

The list **Items** contains any item specified by you. Let us look at the individual properties:

* *ID*: Each item has a unique ID in form of an integer number.
* *Name*: Name of the item
* *Category*: A free category
* *Shop category*: A free shop category
* *Owned*: How many of these items owns the player at the beginning?
* *Can buy*: Can this item be bought in a shop?
* *Buy price*: The price of the item
* *Sell price*: The money you get for selling this item
* *Can own multiple*: Can the player own multiple of these items?
* *Image*: An image file shown for this item.
* *Skill 1-3*: Three blank skills

To add a new item, fill the item box at the bottom of the tab and click **Add new**. To delete an item, select it from the list and click **Delete**. If you double click an item in the list, the item properties are shown in the item box. To update it, select the item to update and click **Update**.

TweeFly generates several Macros for you to manipulate the inventory.

Table 2: Inventory macros

|  |  |
| --- | --- |
| Name | Description |
| initItems | Initializes your items (no need to call) |
| initInventory | Initializes your inventory (no need to call) |
| addToInventory | Adds an item to the inventory. Requires two parameters: Item ID and the amount to be added. |
| removeFromInventory | Removes one or multiple items from your inventory.   * If one parameter is specified: All items with the specified ID (parameter 1) are removed. * If two parameters are specified: The amount n (parameter 2) of the item with a specific ID (parameter 1) are removed. |
| inventory | Renders an inventory table for the paragraph section. All properties marked in the *Display in inventory* list are visible. |
| inventorySidebar | Renders a two-column inventory table for the sidebar with a tooltip giving information on each item when the corresponding checkbox is checked. |

A sample configuration with the inventory link in the sidebar, the inventory itself in the sidebar, four items and five selected properties to display in the table will look like the picture below.

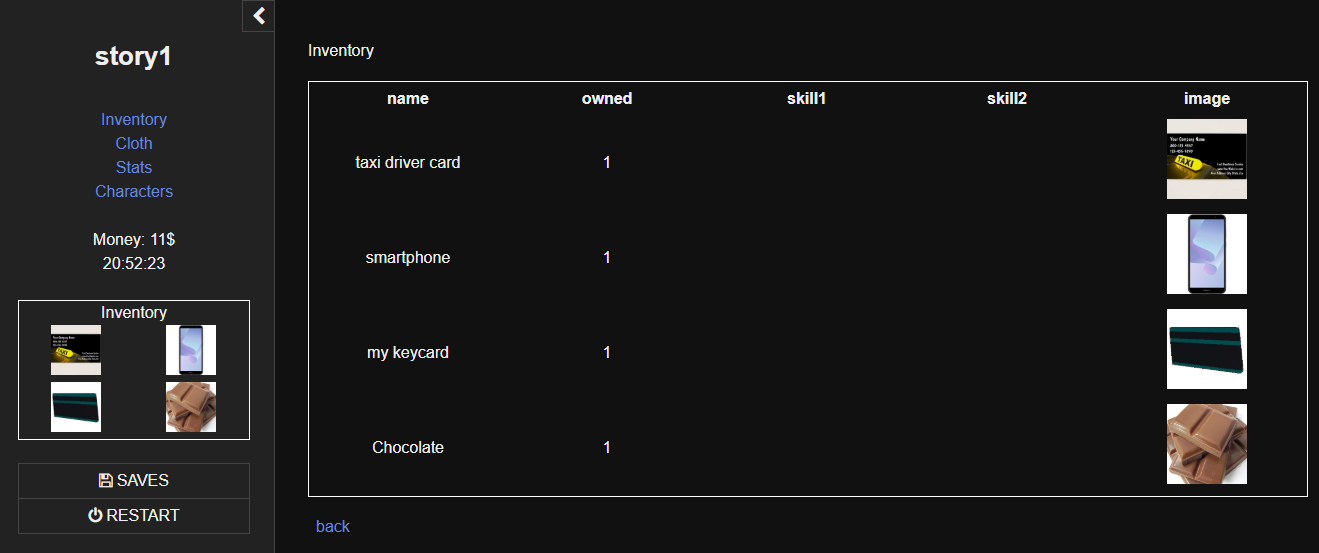


Figure 4: Inventory rendered in sidebar and in paragraph section.

Internally, all items defined (no matter if the player owns the item or not) can be accessed via the list in the JavaScript list **state.active.variables.items**. The items actually owned by the user are stored in **state.active.variables.inventory**.

## Clothing tab

The clothing tab is more complex than the inventory since it makes use of three lists:

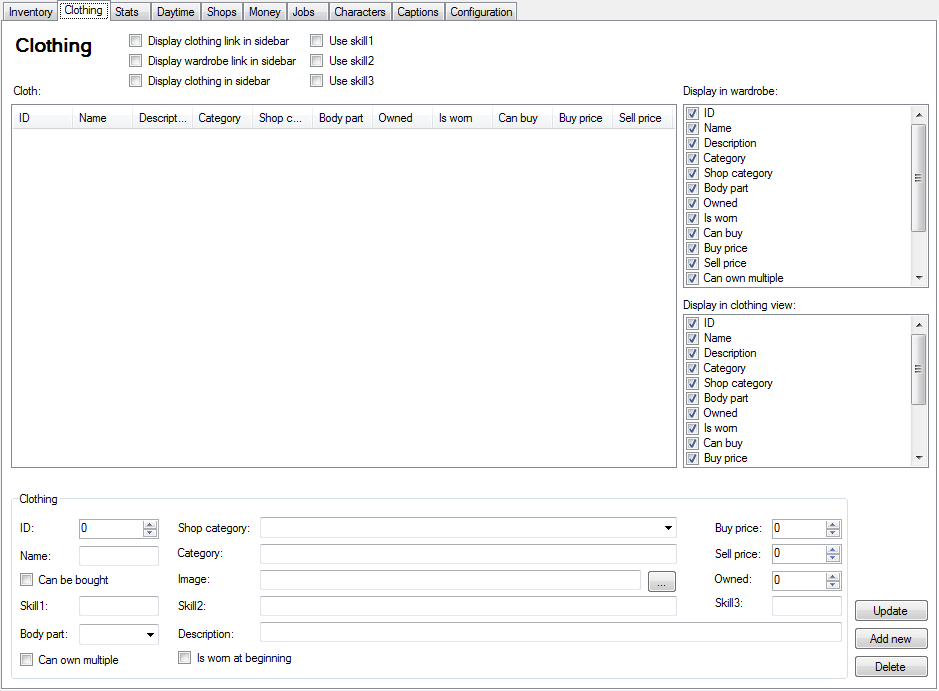
1. A list for all clothing in the game (**state.active.variables.allClothing**)
2. A list for all cloth owned by the player (**state.active.variables.wardrobe**)
3. And a dictionary for each cloth worn at a specific place (**state.active.variables.wearing**)

Similar to the inventory, the clothing tab provides a list of checkboxes on top. **Display clothing link in sidebar** and **Display wardrobe link in sidebar** allow the user to view either the actually worn clothing or the wardrobe whenever he/she clicks the link. **Display clothing in sidebar** creates a table with all clothing worn by the player in the sidebar on the left.

Clothing can have one to three blank skills attached which can be activated checking **Use skill 1-3**.

The definition of clothing can be done in the clothing box at the bottom of the tab. **Update**, **Delete** and **Add new** clothing can be done via the three buttons next to the clothing box. Double clicking an item loads an item into the clothing box.

The two lists on the right (**Display in wardrobe** and **Display in clothing view**) determine which clothing properties are shown in the clothing, wardrobe and clothing sidebar tables.



Clothing has the following properties:

* *ID*: Unique number.
* *Name*: The name of the item.
* *Description*: The item description.
* *Category*: A clothing category.
* *Shop category*: A shop category
* *Body part*: Where this clothing is worn (see the drop down list in the clothing box for possible values).
* *Owned*: How many clothing of this type has the player at the beginning.
* *Is worn*: Is this clothing worn at game start?
* *Can buy*: Can this clothing been bought in shops?
* *Buy price*: How much is this clothing in shops?
* *Sell price*: The money you get for selling this item.
* *Can own multiple*: Can the player own more than one clothing of this type?
* *Image*: An image to display this clothing.
* *Skill1-3*: Three blank skills

The macros used in the clothing system are:

|  |  |
| --- | --- |
| Name | Description |
| initAllClothing | Initializes all clothing in the game. |
| initClothing | Initializes all clothing the player is wearing. |
| initWardrobe | Initializes all clothing in the player's wardrobe. |
| clothing | Prints a table with the clothing the player wears. |
| clothingSidebar | Creates a table of images in the sidebar with the clothing worn by the player. |
| wardrobe | Prints a table with all the player's clothing. |
| addToWardrobe | Adds clothing and expects two parameters: The clothing ID (parameter 1) and the amount to add (parameter 2). |

# Change log