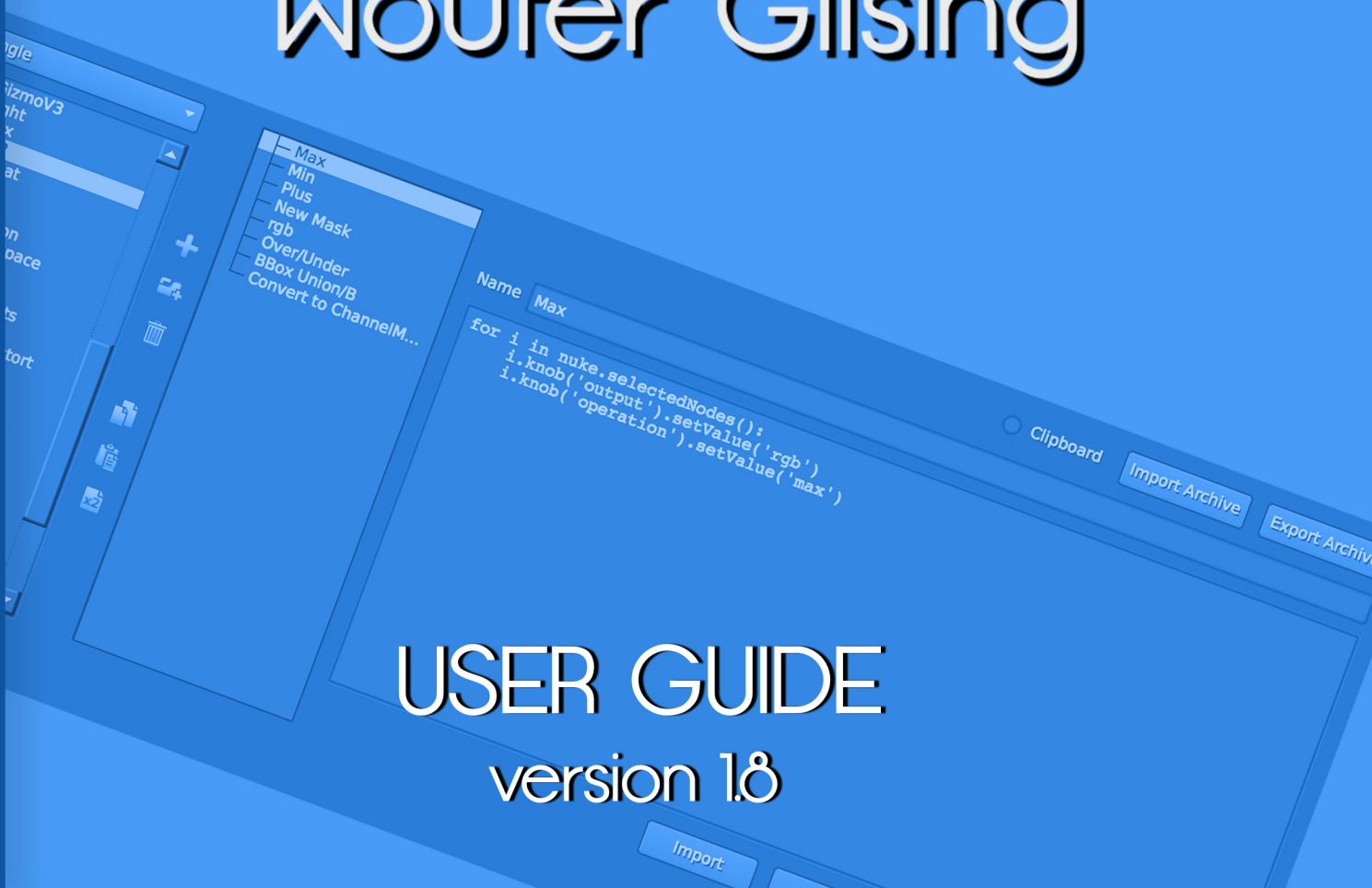

# W_HOTBOX

## Wouter Gilsing




# USER GUIDE

version 1.8

# Table of content

# Introduction

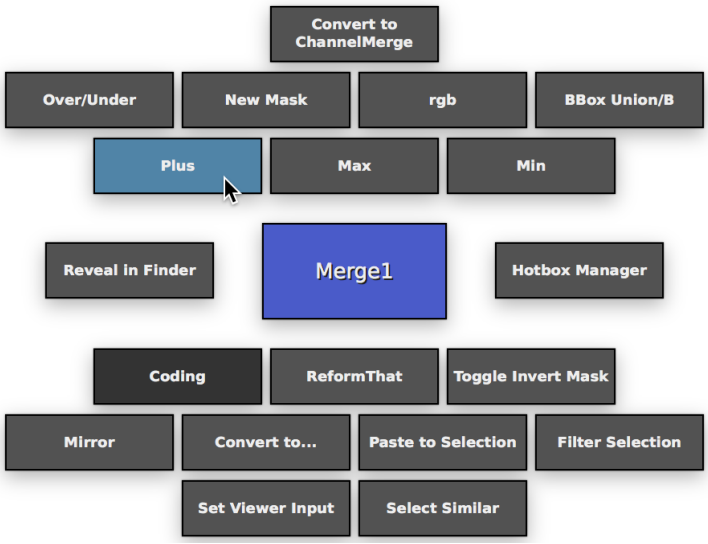
Hi there, thanks for trying out W_hotbox!

In this documentation I'll describe everything there is to know about the features of W_hotbox. It also includes an installation guide, python examples and my contact information. In this first section I'll give a quick introduction of why I started developing this tool to begin with.

It al started with me looking for ways to manage my python scripts better and more efficiently. I especially wanted the scripts to be easier accessible. Before that I was either adding the scripts to menu items or accessing them using shortcuts. Having to browse through menu's kinda contradicted the reason I wrote the scripts in the first place though: speed up my workflow and make things easier. Besides that I felt I was running out of (easy to use) shortcuts as well.

I also wanted to make the process of adding new scrips easier, as I often didn't feel like opening a text editor, saving the script to disk, making the code accessible inside Nuke by adding it to my menu.py and finally having to restart Nuke. Especially small scripts (think of two or three lines of code) although sometimes very powerful, were not really worth saving to a file, and 'wasting' a hotkey on.

To solve these 'problems' I developed what ended up as the Hotbox you just downloaded. Users that are familiar with Autodesk Maya will probably recognise that concept. It's basically a fully customisable 'favourites menu' that pops up for as long as you press the shortcut and disappears as soon as you release. The Hotbox Manager allows you to add new scripts on the fly, which are directly accessible via buttons that appear in the menu under your cursor.

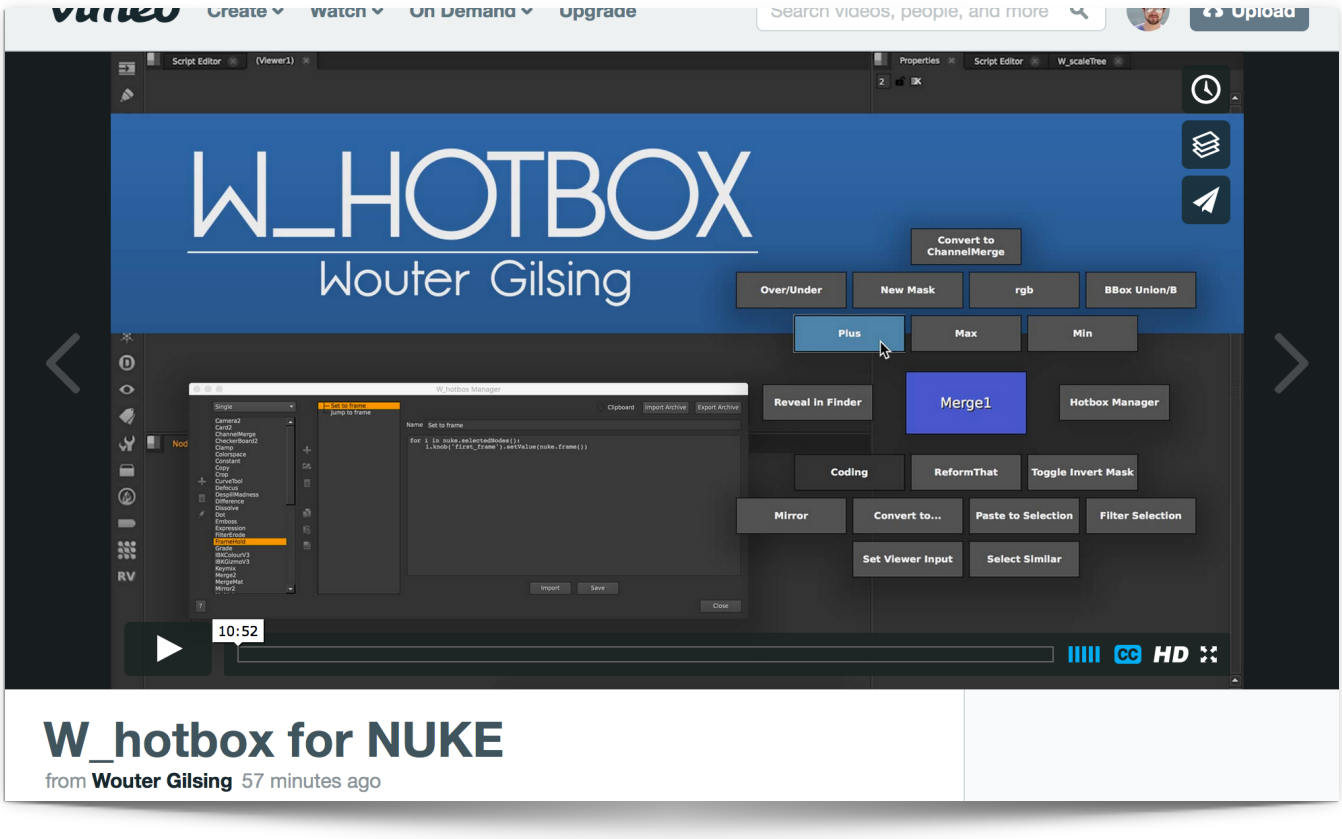



For me it changed the way I interact with Nuke completely. The ability to move any actions to a button that appears near your cursor will save you a trip to the other side of your screen and back pretty often. Besides that I use it a lot to automate repetitive tasks and automate actions I would otherwise probably not have taken time for at all. Think of things that make your script more readable, but don't necessarily contribute to the final image, like (color)labeling nodes.

It's in no way an essential tool, as Nuke functions perfectly fine without it obviously. Therefore using the Hotbox might take a little bit of time to get used to. Besides that it takes a bit of time to add the buttons you want and customize the Hotbox to fit your own needs, as everyone works differently. You'll see though, that when you come up with

an idea for a button while working on a shot, it usually takes no more than a few minutes at most to create that button.

Of course, if you know Python to some extend it will make it way easier to write your own more complex buttons. I think though, that you should still be able to create your own buttons to do simple tasks (like modifying knob values and creating new nodes), when you have no programming skills whatsoever. Just install the (optional) buttons that come with the download and copy and paste stuff around. It often takes no more than changing a knob name in a python script to give a button a totally new purpose. I found that these simple tasks are the ones that I personally use the most anyway.




I would like to make you aware of a video I recorded that gives a quick demonstration of the Hotbox and will show you everything you need to know to get started.

That video can be found here:

https://vimeo.com/woutergilsing/whotboxintroduction

Note that this demonstration was recorded using version 1.0.

If you decide to install the tool in a studio environment, I would love to hear which studio! (Just because I think it's a fun thing to know.)

I hope you like it!

# Hotbox

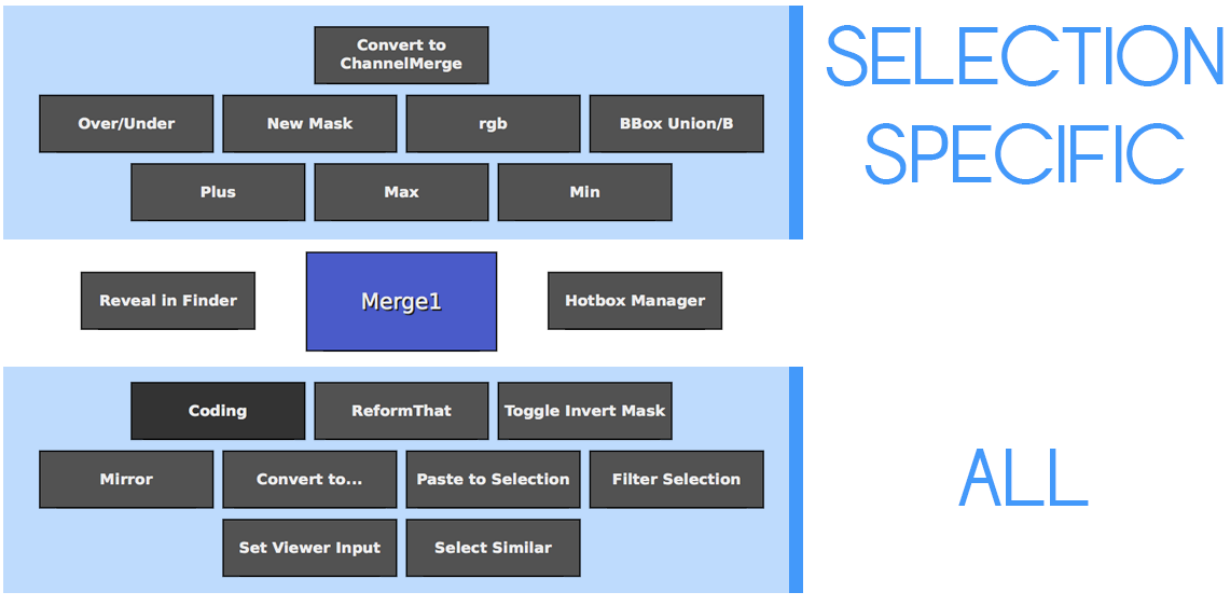
By default the Hotbox is launched when holding the ‘back tick' key (`), the symbol that shares a key with the ‘tilde’ (~) and is located above the ‘Tab’ on most keyboards. This is customizable through the preferences (see the Preferences section of this document for more information).


The Hotbox will only be accessible for as long as the user holds the key and will disappear upon release.

The buttons that show up when launching the Hotbox can be divided in two categories. At the top you'll find the buttons that are selection specific and change according to the selected node classes. The buttons at the bottom half will always be there, no matter what is currently selected. Buttons are activated by clicking.

The bigger button in the centre is the ‘selection indicator’. This button will display the name and color of the currently selected node class.

Buttons that are colored a darker shade of grey are menus, and will open a new set of buttons when activated. When browsing through submenus, clicking the button in the centre will take you back to the root level.

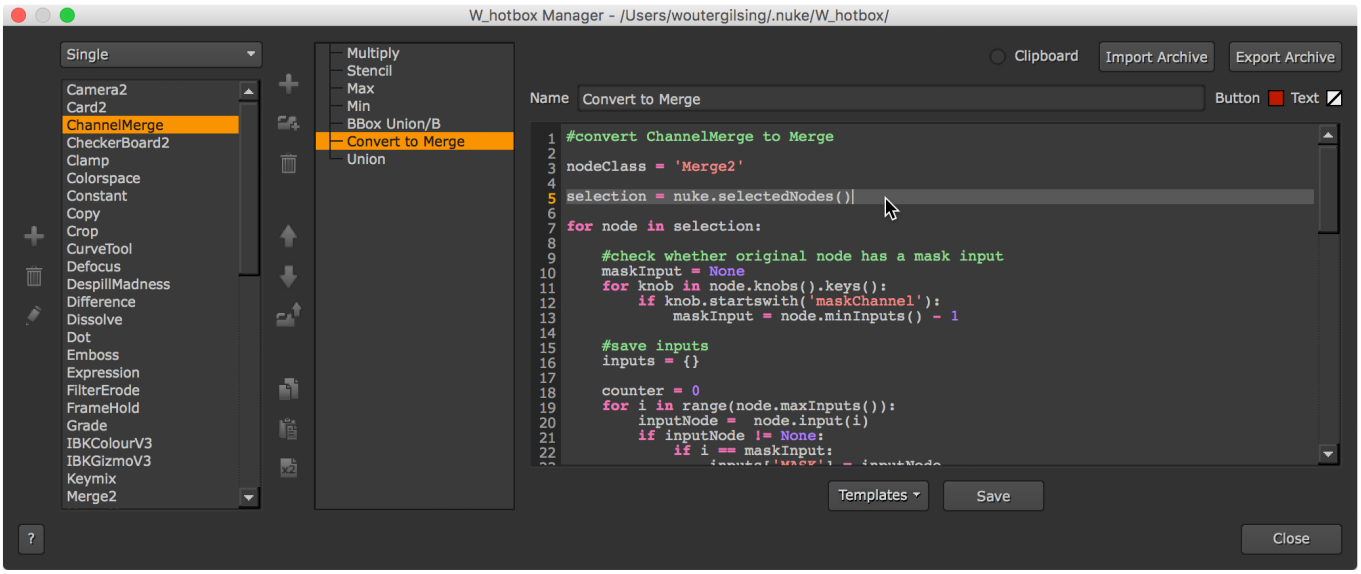


# Hotbox Manager

Without buttons the Hotbox is pretty useless, to add and modify buttons the user can use the Hotbox Manager. To open up the manager, open an instance of the Hotbox and click the button right of the 'selection indicator' button, saying '*Hotbox Manager*'.

Alternatively the Manager can be launched by clicking the *'open Hotbox manager'* button found under the W_hotbox tab in Nuke's Preferences Pane, or by clicking '*Edit/W_hotbox/Open Hotbox Manager'*.

The window that pups up is the manager. The manager can be divided in three sections. From left to right: the classes column, the buttons column and the script editor. From now on items living in the first and second column will be referred to as 'classes' and 'buttons' respectively. The user is supposed to built it's selection up from the left. When selecting a class the button column will no longer be greyed out. When a button is selected the user can make modifications to that button using the script editor.

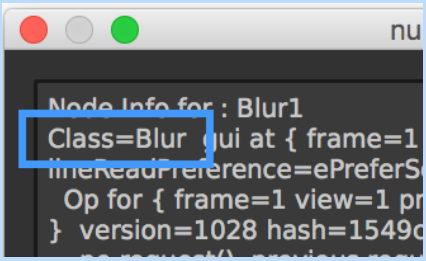


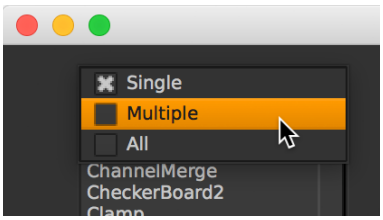
## Adding new classes

In order to make a button show up when a specific node is selected the user should start by creating a new entry for the nodeclass.

To add a new nodeclass simply click the '+' icon to the left of the far left column. This entry should have the exact same name as the nodeclass of the node the user wants to associate with the buttons.

*The class of a node can be found by hitting 'i' with the node selected, while being in the Node Graph. Most of the nodeclasses are pretty straightforward: 'Grade' for a grade node and 'Blur' for a blur node for example. However, some nodes have an other, more complex nodeclass than you would expect. The class of a merge node is called 'Merge2' and 'OFXuk.co.thefoundry.keylight.keylight_v201' is the nodeclass of the Keylight node, to name a few.*




Before adding a class, you should make sure you are in the right mode. Above the column a pulldown menu is located, which displays the current mode of the Manager. There are three modes; Single, Multiple, All and Rules. Where or when a button will appear depends on what mode you add it to.

## Single

'Single' is the most common mode. As the name suggests it deals with a single nodeclass at the time. For example; all the buttons that are added to a class called 'Blur' will appear when the selection of nodes solely exist of nodes of that specific class. It doesn't matter how many nodes are selected at once, as long as they all share the same nodeclass.

### No Selection

*It's also possible to add buttons to the Hotbox that shows up when nothing is selected. This can be done by assigning buttons to a nodeclass called 'No Selection'. This nodeclass will be created be default and can be found in the 'single' mode.*

## Multiple

The Multiple mode deals with buttons that show up when a selection holds multiple different types of nodeclasses. When adding a new class to the classes column, nodeclasses can be separated by using a '-'.

For example: Buttons added to a class called "Transform-Grade-Blur" will show up when the current selection contains at least two of those node classes. Again, similar to the Single-mode,

when a different kind of node is selected as well, the Hotbox won't show any of the buttons added to the "Transform-Grade-Blur" class at all.

### All

The buttons defined in the All section will appear no matter what nodes are currently selected and those buttons will appear at the bottom half of the Hotbox. Since the All section does not take the current selection of nodes in account, the user can start adding buttons right away without having to define a nodeclass first. The class column will therefore stay greyed out in this case.

#### Groups

*Besides being able to add a 'Group' class (by adding a class called 'Group' in the Manager as you would do for any other node) the user is able to define specific classes for groups. In this case the Hotbox will have a look at the name of the selected nodes as well.*

*Currently, groups with a name ending in a number should be added in the Manager with those numbers remove (e.g. 'myCustomSnippet_v13' should be added as 'myCustomSnippet_v'). Numbers at another index of the name won't cause any problems. This only applies to instances of the 'Group' class. Keep in mind the Hotbox will only show up when the name of the group node*

### Rules

**Introduced in v1.8**

Similar to the previously discussed classes, a 'rules' too can be seen as a 'filter' that decided which set show up. However, where classes only let the user pick a nodeclass, the options with rules are fully customisable, as a rule is a python script.

Setting up a rule is similar to setting up a new class. Just click the '+' icon on the left side of the manager. Once you have given your rule a name, you will notice that the script editor now becomes available. This is where you define the actual rule. Similar to Python expressions in Nuke, the rule will communicate its result through a variable named 'ret'. Whenever 'ret' equals True at the end of your rule, the buttons associated with that rule will show up. To give an example of a rule:

```
ret = any([node.name().startswith('B') for node in nuke.selectedNodes()])
```

This rule will look at all the selected nodes and check is the names start with the letter 'B', if this is true for any of the nodes, it will set a variable named 'ret' to True. In that case all the buttons you associate with this rule will show up.

In this case the rule was nicely wrapped up in a single line. However, this is not a necessity. You can make you rule as long as you want. This rule would result in the exact same outcome, just written down in a different way:

```
for node in nuke.selectedNodes():
    if node.name().startswith('B'):
        ret = True
```

In the case of these two examples, the rule deals with attributes of specific selected nodes. This is the most obvious, and probably most useful way to setup rules.

However, a rule can be anything and don't need to be selection specific necessarily. To give some examples; you could set up a rules to have specific buttons only appear on when your cursor in hovering over the viewer, or just on Mondays after 3pm, or only when you are in a specific version of Nuke, or just on a specific OS.

Different than with classes, every rule has a checkbox in front of its name. This checkbox is to give the user the option to disable a rule, without having to fully delete it.

Above the script editor, a checkbox labeled 'Ignore classes' can be found. Checking this box will ensure that, whenever a rule evaluates as true, it won't bother to look at any classes that might have been setup that match the current selection.

## Adding new buttons

After having added and selected a class or a rule, the actual buttons can be managed in the second column.

Clicking the '+' sign left of the column will get you a new button. By default a new button will be called 'New Item'. When selecting that button you will notice the script editor is no longer being greyed out, and you are now able to make modifications to the name and the associated python script of the button.

The script editor is the area where you define the actual python code that get executed when hitting the button. Python-wise you should threat your code exactly the same as you would do in Nuke's script editor.

The button underneath the **+** sign will allow the user to create submenu's. Those submenu's can hold other buttons or submenu's.

The three button in the middle can be used to change the order of the buttons. The first to will simply move the selected button up or down. Depending on the state of a submenu (expanded or collapsed) the button will skip over it or enter the submenu. The last button will move the any selected button out of a submenu.

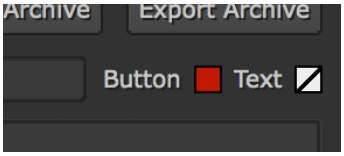
The three last buttons left of the buttons column are to copy, paste and duplicate, respectively. **Duplicate** will create an exact copy of the selected button in the currently selected class. **Copy** and **Paste** allow the user to copy a button from one class to the other. Simply click the **Copy** button to place the button in the manager's clipboard (which differs from the system's clipboard and will be emptied every time the manager is closed), navigate to the class you want the button to copy to and click **Paste**.

# Button Appearance

## Color

Next to the name input field, at the top right of the Manager, two color swatches can be found.

Those two color swatches will respectively control the background color and text color of the currently selected button. Clicking the swatch will cause a color picker to pop up, enabling the user to choose a custom color.




When set to its default color, the color swatch will display a diagonal line.

When enabled the text color will change depending on the color of the button, in order for the button to maintain enough contrast to remain readable. This only applies when the text color is set to its default color.

For example, changing the button's main color to a bright yellow tone, while having the text color set to its default white color, will change the latter to a dark grey instead.

The way the swatch is clicked will affect the action that is triggered. An overview of the shortcuts and their corresponding actions can be found in the table below:

| | |
|---|---|
| LMB | Open the color picker to choose a custom color. |
| RMB | Revert to default color. In case of the swatch controlling the text, this will also invert the color. |
| CTRL + LMB | Paste the color from the clipboard to the color swatch. |
| CTRL + RMB | Copy the color swatch's current color to the clipboard. |
| SHIFT + LMB | Change the color of the color swatch to the color of the currently selected node. |
| SHIFT + RMB | Copy the color of the color swatch to the clipboard, formatted as a 32bit value as used by nuke for interface colors. |

By default, the Hotbox will query the font to use from the nuke preferences to make sure the Hotbox matches the rest of the Nuke interface. However, the appearance of the text inside the Hotbox buttons is customizable on a per button level, as it supports the use of HTML tags.

Before the previously mentioned color swatches were added to the interface (v1.6), the only way to change a buttons appearance was by using HTML tag in the name. This is still possible, and offers you some additional formatting options, besides color.


The figure above shows an example of the must commonly used tags. Those effects can be achieved by surrounding the name of the button with the following code:

| | |
|---|---|
| … | *Italic* |
| … | **Bold** |
| <s>…</s> | ~~Strike through~~ |
| <u>…</u> | Underline |
| <font size = “15”>…</font> | Size |
| <font face = "Comic Sans MS”>…</font> | Font Family |
| <font color = “Red”>…</font> | Color |

Colors can be assigned by using either the name of the color (like in the example given, “Red”) or as a hexadecimal number. A detailed list of color names and their corresponding hexadecimal number can be found here: https://en.wikipedia.org/wiki/X11_color_names#Color_name_chart
When the color swatches are set to a non-default color, this will overwrite the HTML color tag.

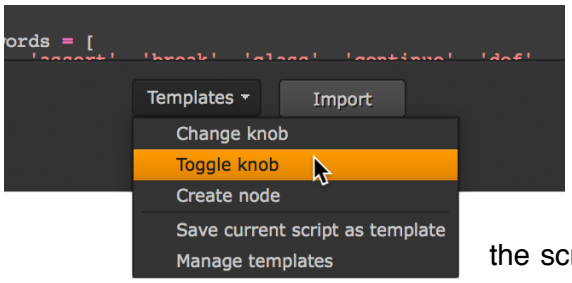
However, since the introduction of the color swatches, this method has become inferior.

Images are also supported by using the following tags

```
<img src=“path/to/image“>
```

The buttons have a resolution of 105 x 35 pixels. When assigning an image of a different resolution, no scaling will be applied.

# Templates




Templates allows the user to save snippets of code to be quickly accessedat a later point in time. This can be useful for commonly used code like the code required to change the value of a knob on the selected node.

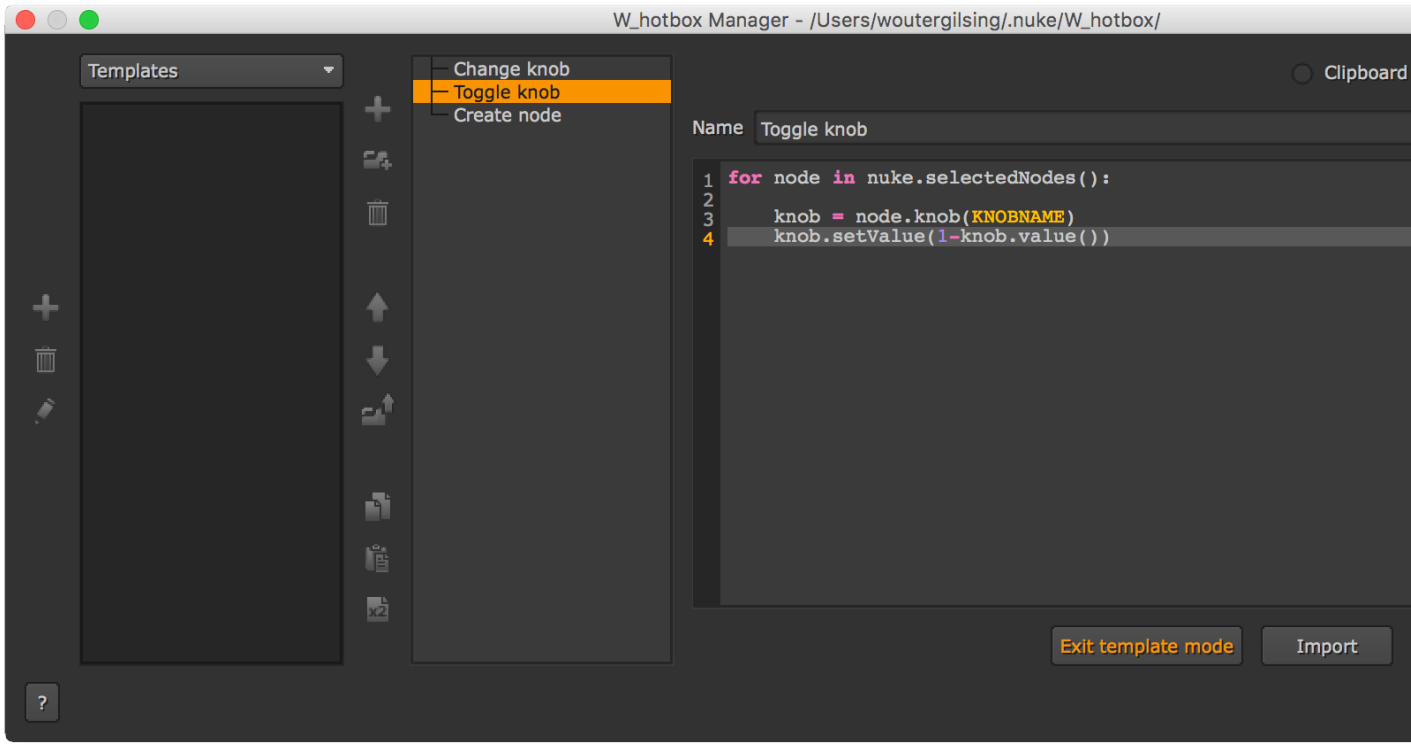
Anything having todo with those templates can be found in the template menu. This button can be found underneath the script editor and clicking it will reveal a dropdown menu. Every menu entry above the divider line symbolises a template and will, when clicked, insert it's template script into the script editor. Loading a template won't overwrite any existing code, unless part of the original code was selected.

To create new templates or manage existing ones the user needs to enter the Manager's 'Template mode'. This mode can be activated by clicking 'manage templates' at the bottom of the dropdown menu.

The usual 'Single/Multiple/All' dropdown of the Manager is now set to 'Templates'. All 'buttons' created in Template mode, will show up as menu entries in the template dropdown menu. Similar to the way regular buttons work, templates can be stored in submenu's as well.

There are several keywords functioning as placeholders that will be color coded bright yellow when being typed into the script editor. Those keywords are KNOBNAME , NODECLASS, NODENAME, VALUE and EXPRESSION. Those keywords are by no means required to make the template function properly.

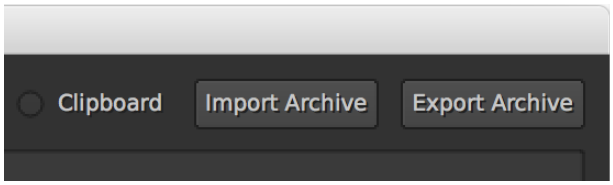
To return to regular the Manager the user can click the 'Exit template mode' button that appeared at the spot where the template dropdown menu used to sit.



# Exporting and restoring settings

To easily share settings between different machines, the user can choose to export an archive file to disk. When clicking the ‘Export Archive’ button the user will be prompted to select a location where the file will be saved. This .hotbox file can be reimported by hitting the ‘Import Archive’.

Alternatively, the contents of the contents of the .hotbox file can be directly imported from, or exported to the systems clipboard. Simple tick the ‘Clipboard’ bullet next to the two import/export buttons. The contents that are copied to the clipboard are encoded and will be decoded when imported.



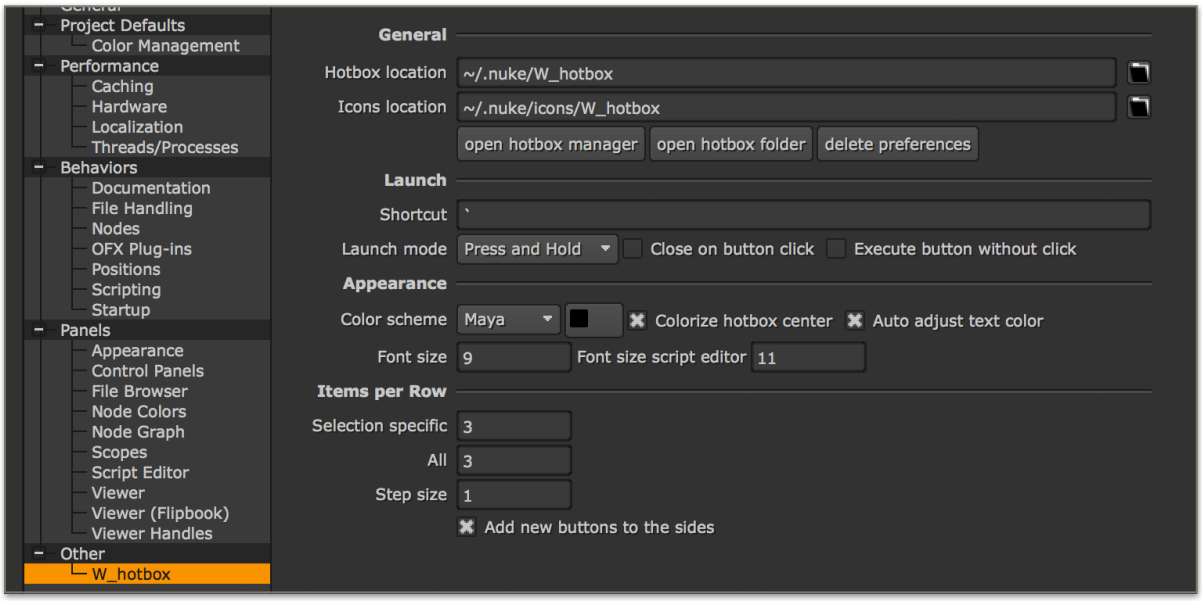

When importing an archive, the currently existing archive will be appended. Any buttons with the same name will be replaced. Submenu’s of the same name will be merged.

Currently importing an archive generated on a machine running OSX or Linux won't work on a machine running Windows **when using the clipboard**. However, sharing buttons by writing them to an archive file will work as expected.

Importing archives using the clipboard on a machine running OSX or Linux will work as expected.

# Preferences

When Nuke is launched for the first time after installing the Hotbox, several knobs will be added to Nuke's preference panel. These knobs can be found under the ‘W_hotbox’ tab.




## Location on disk

The first two knobs, **‘Hotbox location’** and ‘**Icons location**’ have to do with the installation of the Hotbox. By default the Hotbox will be installed in the .nuke folder located in the users home directory. If the user would rather pick a different location, he can define so here.

## Launch

**‘Shortcut’** defines the keystroke that needs to be triggered in order to launch the Hotbox. As of right now, the user is only able to change the shortcut to a single keystroke without a modifier key. After changing the shortcut, Nuke has to be rebooted in order for the changes to take effect.

The **‘Launch mode**’ dropdown defines the way the Hotbox is launched. The user can choose between pressing the shortcut and holding it down, and activating it with a single tap. When set to ‘Press and Hold’ the Hotbox will disappear as soon as the user releases the key. When set to ‘Single Tap’ the shortcut will toggle the Hotbox on and off.

When **‘Close on button click’** is enabled, the Hotbox will close whenever the user clicks one of the buttons (excluding submenus obviously). This option will only take effect when the launch mode is set to ‘Single Tap’.

**‘Execute button without click**’ will execute the button underneath the cursor upon closing the Hotbox. Clicking a button will still work. This behaviour is recommended when using a trackpad.

## Appearance

The '**Color scheme**' dropdown let's the user have the option to change the highlight color of the buttons. By default this knob is set to 'Maya' to mimic Autodesk Maya's muted blue tones. Alternatively the user can pick 'Nuke' for a bright colored orange, or 'custom'. When going with 'custom' a custom color can be chosen using the color tile next to the dropdown menu.

By default the center button of the Hotbox will change colors depending on the current selection. When leaving **'Colorize Hotbox center'** unticked the center button will be colored a lighter tone of grey.

**'Auto adjust text color'** defines whether or not a buttons' text color should be changed automatically whenever a user changes the base color of a button. This is done to try to keep the buttons as readable as possible. Changing the a buttons' base color to a bright yellow will change the text color from white to black, in order to remain some contrast. This only happens when the text color is set to its default color. In case the text knob is adjusted as a result of a changed base color, this new color will still count as 'default', so changing the basecolor back to a darker color will make the text pop back to white again. Default colors won't be saved to disk, so in case the user made alterations to the base color whilst leaving the text color set to default, switching this option on or off will likely change the appearance of the Hotbox

There are two knobs dealing with font sizes. **'Font size'** will give the user control of the size of the text that appears in the Hotbox buttons (this will be overruled in case by a custom font size set by the user on a per-button level using HTML tags).

**'Font size script editor'** will allow the user to change the size of the text displayed in the editor area of the Hotbox manager.

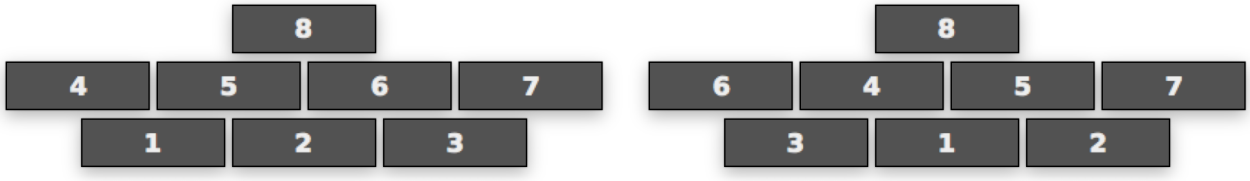
## Items per row

The **'Selection specific'** and the **'all'** values represent the maximum amount of buttons a row can contain (in respectively the upper and lower half of the Hotbox). When a row's maximum capacity is reached a new row will be started. This new row's maximum capacity will be incremented by the step size, resulting in the triangular shape when having multiple rows.

When **'Add new buttons to the sides'** is enabled, new buttons will be added left and right of the row alternately, instead of to the right.

In the image below you can clearly see the difference between the two modes (whereas left is disabled and right is enabled). The '1' button in the middel of the bottom row, was located exactly in the middle when there where only one button. As you can see in the left image however, when this user added more buttons, it got slowly pushed more towards the side and ended up at the very left of the row.

In the right image the option was enabled. This way any existing buttons will hardly change position when new buttons are added, taking muscle memory in account. This function is turned on by default.



# Working in a studio environment

Working in a studio environment might have some special requirements, like being able to have multiple repositories to draw buttons from. Think of a button set that changes per project, a facility set, or both.

## Adding a new repository

Adding an additional repository can be done by adding two environment variables, called 'W_HOTBOX_REPO_PATHS' and 'W_HOTBOX_REPO_NAMES'. The first one is used to define the desired locations and the second one will store the names of those repositories (think of the name as a description, for example SHOW or STUDIO) You can add multiple repositories, by separating the entries with a ':' (Linux and OSX) or a ';' (Windows)

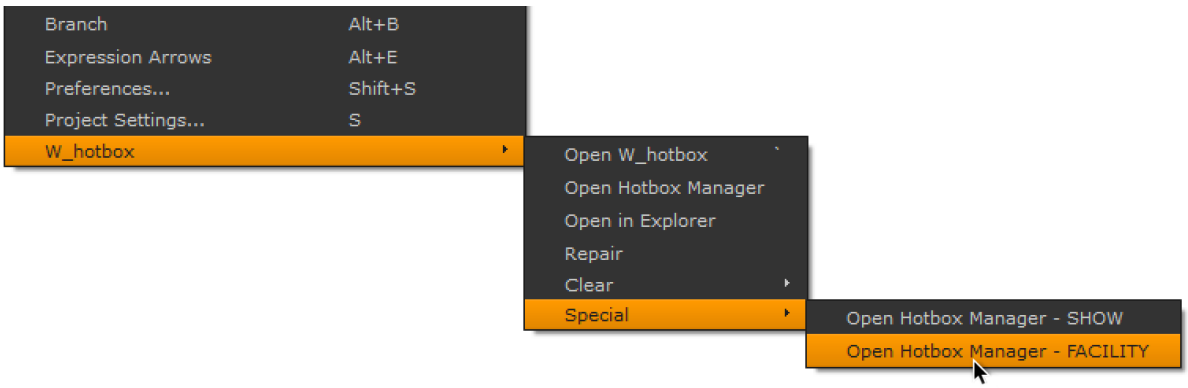
```
W_HOTBOX_REPO_PATHS=/path1:/path2:/path3
W_HOTBOX_REPO_NAMES=name1:name2:name3
```

Make sure to point the path(s) to the folder **containing** the 'All','Multiple' and 'Single' folder. Every repository that is added should have a unique name and an unique location.
Also beware that for every path that you add, a name should be added as well (and vice versa), otherwise that repository will be ignored. Path1 corresponds with name1, path2 with name2 etc. You can add an unlimited amount of repositories. Just keep in mind the Hotbox accesses these folders every time you hit the shortcut, so adding 500 extra repositories might affect the working experience in a negative way.

Buttons loaded from one of the additional repositories will be outlined grey, rather than the usual black.

## Repository specific Hotbox Manager

When launched from the Hotbox, the Manager will still only show the set of buttons that was added to the default repository (the one defined in the Nuke preferences). To edit the buttons that live in one of the extra repositories the Manager can be launched from the menubar. All the added repositories will show up under *'Edit/W_hotbox/Special/Open Hotbox Manager - REPOSITORY NAME'.* If no extra repositories were added, this 'special' menu item won't show up at all.



## Disabling the knob to change the icons location

By using the ‘icon location’ knob found in the preferences, the user can change the location of the icon that are used to populate the Hotbox Manager. Setting an environment variable called ‘W_HOTBOX_HIDE_ICON_LOC’ to ‘true’ hides this knob from the preferences panel. This way artists aren’t able anymore to manually pick a location for the icons the are used for the Hotbox Manager.

# Installation

## Clean installation

1 Copy '*W_hotbox.py*' and '*W_hotboxManager.py*' to a folder that's part of the nuke plugin path.

2 Append menu.py with the following code:

```
import W_hotbox, W_hotboxManager
```

3.a Copy the folder named '*icons'* to your .nuke folder. (If you would rather place the icons elsewhere, make sure to follow step 3.b. If you decide to put them in the default location, you can skip step 3.b)

3.b Launch Nuke. Open the Preferences Panel and navigate to the W_hotbox tab. change the path pointing to the icons folder to the folder you placed the icons in step 3.a.

Step 4 is optional. The download ships with a set of buttons ready to use with the Hotbox. The Hotbox will function fine without those, but the user has to add buttons himself before the Hotbox becomes useful. The buttons that ship with this download are easily removed if the user would rather start from scratch himself.

4 Open the Hotbox Manager by either:
- Launching it from the Hotbox itself,
- Clicking Edit/W_hotbox/Open Hotbox Manager,
- Choosing 'open Hotbox manager' from the preferences panel.

Click the button saying 'Import Archive' at the top right of the Manager, while making sure the 'Clipboard' knob next to it remains unchecked. A file browser appears. Navigate to the file called *'buttonBundle.hotbox'* that came with the download and hit 'open'.

## Upgrade from an older version

To upgrade the Hotbox simply replace the old '*W_hotbox.py*' and '*W_hotboxManager.py*' with their updated versions.

New buttons were added to the Hotbox Manager in version 1.6. Make sure to copy all the icons.

## Deinstallation

1 Open the Preferences Panel and navigate to the W_hotbox tab. Click the button saying 'delete preferences'.

2 Make sure to manually remove all the files that were manually added during the installation.

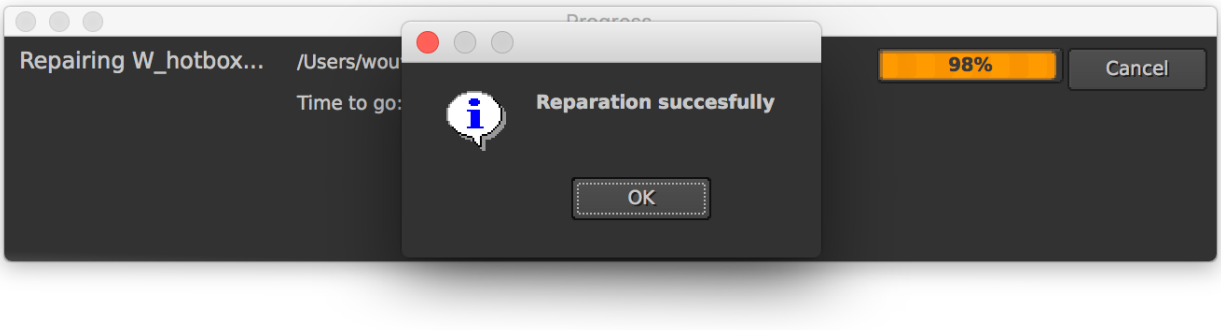
4 Remove the code that was added to menu.py.

3 Inside the .nuke folder a folder called 'W_hotbox' was created. Delete this folder.

# Troubleshooting

When the Hotbox is not working as expected, the first thing that can be done is running the repair tool.

This tool can be found in the menu under *Edit/W_hotbox/Repair*. This will loop through all




the currently existing buttons and will make sure everything is named the appropriately.

If this did not fix your problem, the user can have a look under the hood. The Hotbox folder can be opened in several ways.

- Launching it from the Hotbox itself, by clicking the button saying 'Open in File Browser/Explorer/Finder'.
- Clicking *Edit/W_hotbox/Open Hotbox Folder*,
- Choosing '*open Hotbox folder*' from the Nuke preferences panel.

Rather than deleting them all individually by clicking the bin icon in the manager, it's possible to erase the buttons of the Hotbox in bigger amounts as well, You can do this by using the cleaning tools that can be found under *Edit/W_hotbox/Clear*. It's possible to either delete everything or once ('Clear everything') or by section ('All', 'Multiple' or 'Single').

The Hotbox was built for Nuke 9 and newer.

# Examples

This part of the documentation is targeted towards people without proficient knowledge of Python. It won't touch on any Hotbox specific features, but just will provide examples of very commonly used snippets of code that can be copied and modified to create very useful actions to assign to a Hotbox button.

# Example 1

## Changing a node's knob's value

*Goal: To create a button to quickly change the values knobs of the currently selected nodes using the Hotbox. In this example we will focus on the Merge node and change the knobs called 'operation' and 'output'.*

The following code will iterate over the selected nodes. When writing scripts for Hotbox buttons it's always a good idea to include this loop, in order to make the script work when having multiple nodes selected (even if you're not planning on using the button on multiple buttons t the same time, it won't hurt either). For more information on this subject, you can search for 'for loop python' on google or youtube, as there are lot of great, free resources.

```
for i in nuke.selectedNodes():
```

For every node that's encountered the script will first set the value of the knob named 'output' to 'rgb' and when that's done value of the knob named 'operation' to will be changed to 'min'.

```
for i in nuke.selectedNodes():
    i.knob('output').setValue('rgb')
    i.knob('operation').setValue('min')
```

*The name of a knob can be found by hovering over it in the properties bin.*

# Example 2

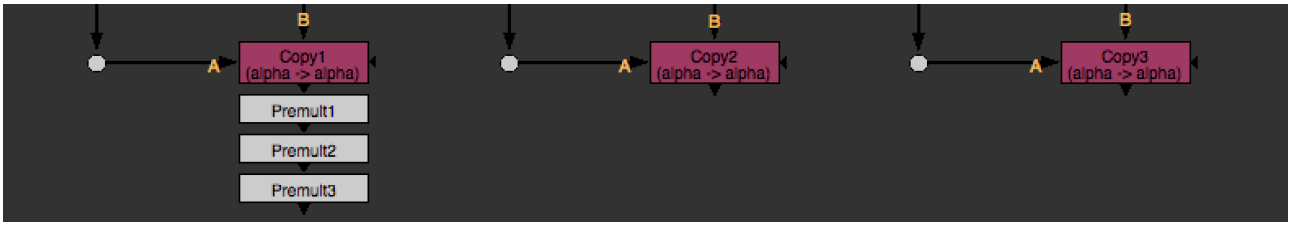
## Creating new nodes

*Goal: To create a button to create a Premult node for the currently selected nodes using the Hotbox. In this case we'll assign this button to the Copy nodeclass, as it is pretty common to create a premult node after a creating a copy. For the sake of this example, we will also change the channel knob from its default 'rgb' to 'all'.*

Similar to Example 1 we could use this as our code:

```
for i in nuke.selectedNodes():
    premultNode = nuke.createNode('Premult')
```

This will do the job if we only have one node selected at the time. However, if we would have more than one Copy node selected at the same time, this code would break and result in something unwanted similar to what is shown in the image below.




Because 'Premult1' will become selected after its creation, 'Premult2' will be connected and placed underneath 'Premult1'.

To solve this problem, every time a new Premult node is about to be created, the Copy node that new Premult node is supposed to connect to needs to be selected.

Since there's no built-in python command to deselect every node, we'll start off by writing our own.

```
def emptySelection():
    for i in nuke.selectedNodes():
        i.knob('selected').setValue(False)
```

Now we can move on to the part of the code that will take care of the creation of the new nodes.

```
for i in nuke.selectedNodes():
    emptySelection()
    i.knob('selected').setValue(True)
    premultNode = nuke.createNode('Premult')
```

As we don't really want the last Premult node that was created to stay selected at the end, we can call our deselection function once again.

```
emptySelection()
```

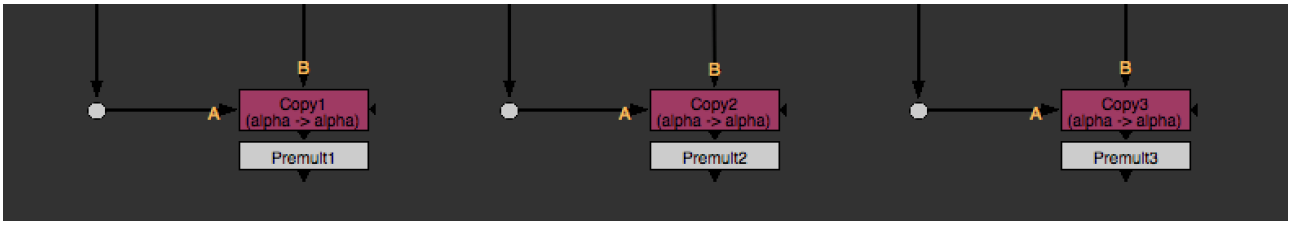
This is what the completed code would look like:

```
def emptySelection():
    for i in nuke.selectedNodes():
        i.knob('selected').setValue(False)

for i in nuke.selectedNodes():
    emptySelection()
    i.knob('selected').setValue(True)
    premultNode = nuke.createNode('Premult')
    premultNode.knob('channels').setValue('rgb')

emptySelection()
```

As shown in the following image, this will work as expected:



# Change Logs

## V1.1
**29 Aug 2016**

Features:

- Added the option to have multiple repositories to store buttons in. Also, the Hotbox manager will now display the path of the loaded repository at the top of the window. Buttons loaded from an additional repository will appear outlined grey, rather than black. See the newly added chapter 'Working in a studio environment' for more information.
- Option to hide the 'iconsLocation' knob from the preferences panel, so artists won't be able to change it, when installed facility-wide.
- When importing an archive of buttons, the buttons will now append the current set, rather than replacing it.

Fixes:

- Version 1.0 did not delete all the knobs added to the preferences panel correctly after clicking 'Delete Preferences', and therefore making the Manager unavailable after a reboot.
- Several minor fixes/improvements to the set of buttons that comes with the download.
- When having other folders in the Hotbox location, only the 'Single','Multiple' and 'All' folders will be affected when a new archive is being imported.

## V1.2
**30 Aug 2016**

Features:

- Improved the way of defining additional repositories (feature added in v1.1). Rather than changing the actual python files the repositories can be defined by setting environment variables. (See page 13 for more information)
- Same applies for hiding the 'iconsLocation' knob from the preferences panel.

Fixes:

- Minor fixes to the set of buttons that comes with the download.

## V1.3
**6 Sept 2016**

Features:

- Knob added  to the preferences panel to control the Hotbox's font size.
- License added

Fixes:

- Knob formerly called 'iconLocation' renamed to 'hotboxIconLocation'.
- Improved the way archives will get created when none is present.

## V1.4
**16 Nov 2016**

Features:

- Error catching. Whenever executing a Hotbox button causes an error, the problem and it's corresponding line will be printed.
- Improved script editor. The script editor of the Manager now includes line numbers, syntax highlighting and auto indentation to make writing code easier. Tab's will be automatically registered as four spaces.

Fixes:

- The Hotbox will now function properly in combination with nodes inside groups.
- Corrected a typo concerning the name of the environment variables in the example on page 13 of this documentation.

## V1.5
**14 Dec 2016**

Features:

- In the script editor, the background color of the selected line now reflects the current state of the loaded script (black - unchanged, white - modified, green - just saved).
- Added the option to launch the Hotbox with a single tap, instead of having to keep the shortcut pressed. This mode is available through the 'Launch Mode'-dropdown in the preferences.
- Reorganised the preferences panel and assigned tooltips to all its knobs.

Fixes:

- Previously resetting the preferences when the Hotbox was recently updated from an old version caused an error.
- Added a section to the manual about how to work around an OSX related bug causing the 'Start Dictation' item to be added multiple times to Nuke's Edit menu.

## V1.6
**17 Apr 2017**

Features:

- The option to change the colors of the text and background of a button.
- Templates. The ability to save snippets of code to quickly access at a later point in time.
- No click execution. Execute the button underneath the cursor upon closing the Hotbox.
- New button order system.
- Auto save. No need to click the 'save' button anymore as changes will automatically be save.
- Option to change to Manager's font size.
- Tooltips added to Manager.

Fixes:

- Manager only floating on top of Nuke.
- Differentiate between groups better/ added the possibility to show buttons for multiple node combinations at the same time.
- Transparency on Linux (no need for the Desktop effect trick as mentioned in the documentation of earlier versions).

- Properly restore selection after certain actions.
- Upper/Lower casing wont affect sorting of classes.
- An empty archive folder used to be left behind when cancelling an archive export.
- Stability improvements.
- Nicer about window.

## V1.7
**27 June 2017**

Features:
- Added support for PySide2 (Nuke 11).

## V1.8
**22 April 2018**

Features:
- Setup custom python rules to have more control over when button appear.

Fixes:
- Now able to run on Nuke 11 Linux.

# Contact

If you encounter any bugs, you can report them by sending an email with a description of your problem to woutergilsing@hotmail.com.

Please mention 'W_hotbox' in the subject of the email and include the following:

- Version of the Hotbox (this can be found in the 'About dialog' which can be accessed by clicking the '?' button in the Manager.)
- Version of Nuke
- Version of your operating system.

Simply clicking the email address in the 'About dialog' will automatically compose an email with all these specifications filled in.

I can't promise anything, but I will try my best to resolve any problems.

Feedback and feature requests are highly appreciated.

# License

Copyright (c) 2016, Wouter Gilsing
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Redistribution of this software in source or binary forms shall be free of all charges or fees to the recipient of this software.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.