NGPS Built-in Applications & Space Conventions

The greatness of the engine is brought by the amazing possibility to interact with it through small modular apps. The built in apps make the engine and their interdependence must be carefully designed. Here we present how these apps interact with each other and through the engine.

Annex 1. Built in NGPS apps [(L) – local app]

1. zoom – provides a user interface to zoom the root camera
2. userMsg – main popup message provider
3. edit – main edit interface
4. edit/components/quickAddInterface – a ring configurable interface that triggers on click and allows various other apps to be launched targeting the element that was clicked
5. editor/components/link – an app that enables the user to link two containers together
6. editor/components/linkEdit – supports link delete / editing
7. editor/components/pchange – allows a container to detect then it overlaps with a sibling or is outside the boundaries of the parent allowing the user to change the parent of the container to the sibling or to the grandparent.
8. editor/components/text – supports addition of text
9. editor/components/sizer – supports sizing and rotation of container
10. editor/components/addImage – supports adding of images in containers ( triggered by quickAddInterface )
11. editor/components/addVideo – supports adding video incontainers ( triggered by quickAddInterface )
12. editor/components/aligner – detects what containers are close to each other & related and displays aligning lines + implements magnet lock
13. editor/components/appChoice – allows the user to load and unload apps and see what apps he/she owns
14. UIexplainer – an application that shows a message explaining a UI button / component on mouse hover or touch

It is important that main editing interfaces do not overlap or remain active while a different interface is active (allowing the user to perform illegal operations and not ensuring exclusive access to the object).

To prevent this the edit app provides a model to which all its subcomponents and other Editor UI apps have to implement:

The main edit app creates a global variable

***Editor*** = {}

***Editor.mainActiveUI*** = {} - is where the hide and activate functions are

***mainActiveUI*** contains:

***hide()*** – a method to be called to close the current active UI

***activate( descriptor, context, activate)*** – activates a new main UI and sets it as current, also hides previous UI. Context – the running context of the functions you pass in the descriptor. Activate – is a Boolean that signals if you want the function to also call the activator method provided in the descriptor.

***descriptor { –*** is an object with the following attributes

***activate*** - function, if you define this when the next activate will be called, this descriptor will be saved and upon the next hide it will call this function ( used to switch back to an active UI after closing the current one )

***passToActivate –*** array of parameters to pass to activator ( if you don’t provide a array [] it will not pass the arguments )

***hide*** – function to call to hide the UI

***passToHide –*** array of parameters to pass to activator ( if you don’t provide a array [] it will not pass the arguments )

***}***

Interdependent apps: some apps may depend on others to perform certain functions. For this to happen the apps need to be globally accessible somehow. The convention is to add the app’s running context (this) to the Editor global object under the same name as the application.

Example: quickAddInterface should be listed as Editor.quickAddInterface

UI Explainer – to implement

* Depends on the regionalization system
* Sets the container’s DOM reference ***data-about*** tag to the explaining text in the appropriate language
* Listens for ***mouseover*** on the DOM object, or in case of touch system, has a trigger that replaces the tap event with it’s own listener (temporarily)