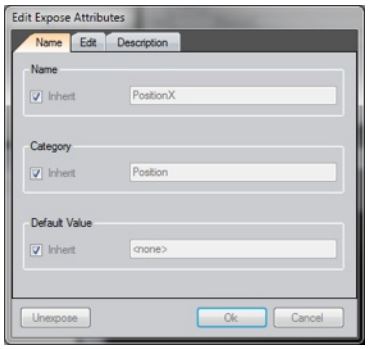
R5.24 Interfacing and Navigation

Property Editor has a Context Menu (RMB click-and-hold on a property)

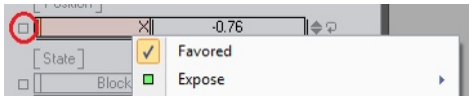
The Context Menu provides access to all old and new functions of the Property Editor.

Favored Only available for Content Editor Selections. Toggles the "Favored" state of a property. Only Favored properties are displayed in Hierarchy Selections and are merged with other properties from other bound nodes. The little vertical bar at the left hand side of the property name indicates whether a property if favored or not. A LMB double click on that bar also toggles the favored state.

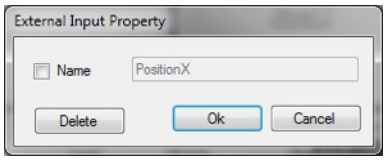
Expose Provides access to all expose functions including "Expose to Interface" (see below) Expose…/Edit… will open the Expose dialog to create and/or edit a property exposing. This dialog has also the option to "unexpose" a member. Each option is checked to inherit the attribute from its actual origin, but you can overwrite all attributes when unchecking the "Inherit" option.



To quickly expose a property without changing any attributes you still can simply click on the small expose indicator. A double click will expose the property and open the expose dialog. A click with the right mouse button will "unexpose" a member quickly and easily.

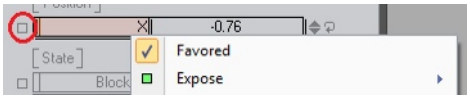


Externalize Provides access to all externalize functions of the Property Editor. Edit… will open the Externalize Dialog. This dialog has the option to "delete" an externalization.



The external name is automatically inherited from its original property, but you can customize the name by checking the Name option. To quick-externalize a property without customizing the name you still can click on the small expose indicator while holding the Control-Key. A

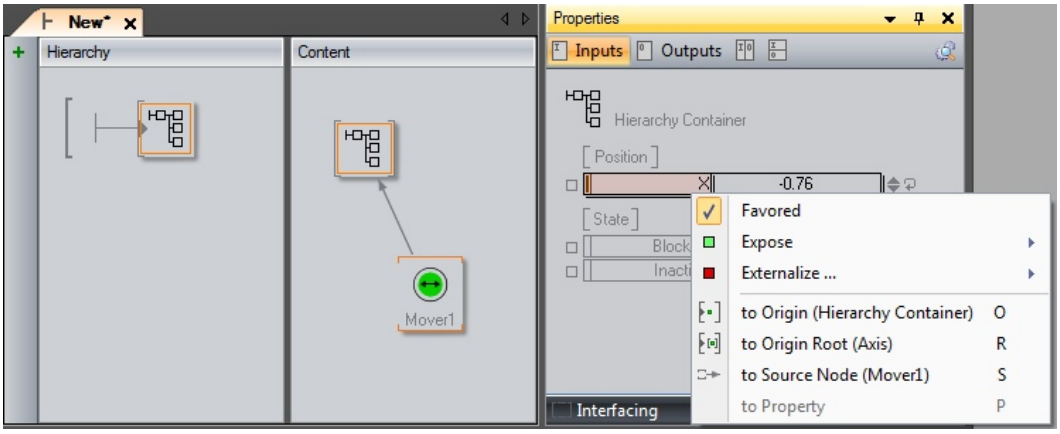
double click will externalize and open the externalize dialog. A click with the right mouse button will delete the externalization quickly and easily.



Please note that a property can either be ■ exposed or ■ externalized but never both!

Property Navigation and Origin Track-Down

The Property Editor often displays properties that do not really belong to the selected node or it displays customized exposed properties of custom containers. To enhance usability in such situations and allow for tracking down the “real” origin of such properties we implemented new navigation functions.



To Origin (O) Navigate to the node which is exposing this property. Usually this node is one level underneath the current level. This name of the origin node is displayed in the menu entry.

To Origin Root (R) Navigate to node which represents the root of this property if exposed across multiple container levels. This name of the origin root node is displayed in the menu entry.

To Source Node (S) Navigate to the node which acts as the value source of the current property (or method). This name of the source node is displayed in the menu entry.

To Property (P) Navigate to the node which owns the current property (or method, or event). If you select nodes in the Hierarchy Editor, you see a merged view of all properties representing the current selection. In our example you will see the Min and Max properties of the Mover

instead of the target property (PositionX). Using this navigation will “reveal” the actual owner of the property and select it in the Content Editor. This name of the owning node is displayed in the menu entry.

- To use the keyboard shortcuts, you have to ensure that the Property Editor is selected (owns input focus and window title bar is highlighted) and the mouse pointer must hover over the property to be navigated. Then press the navigation shortcut O, R, S, P. If the desired navigation is not possible no notification is emitted.

- If navigation points into a sealed container, the next higher unsealed container level will be selected.

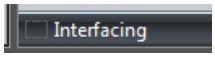
Interface Technology

This groundbreaking new set of features in Ventuz will drastically enhance collaborative workflows, reusability and modularity.

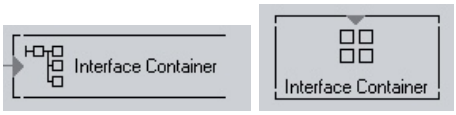
What is an interface? An interface is the exact description of communication between a “user” and “provider” of functionality. The functionality providers can either be a container or an entire scene. The users are other containers or other scenes. The big advantage of having a defined interface between these two peers is that the providers can easily be exchanged against another one – providing another version of the same functionality. Connections (bindings) to the user remain existent, even if the provider is changed or no provider exists at all!

Let’s start with the Interface Containers. To create an interface you always have to work from inside to the outside: create a container node (hierarchy or content container), dive into it and develop the functionality. When done, expose all the properties the user should be able to access. You can provide all types of input and output properties as well as methods and events. It is highly recommended that you provide proper property names, categories and descriptions to these exposed items. Navigate one level up and select the container – you’ll see a normal container with your customized properties.

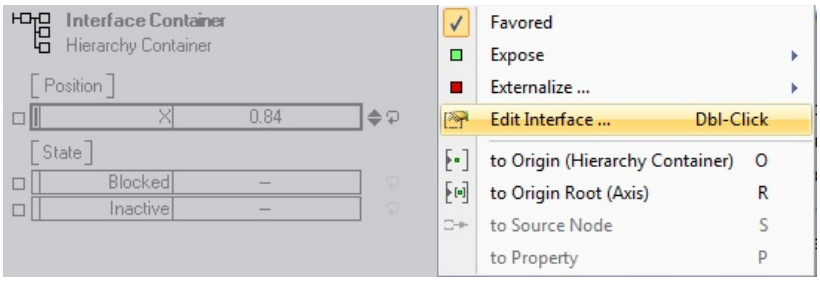
If you expose the “Value” property of the value nodes (float, integer, color, etc) you will see an input property at the container. If the “Value” property is bound to a source node inside the container it will appear as an output property of the container.

If a container is selected in the Property Editor, you’ll see a new button: 

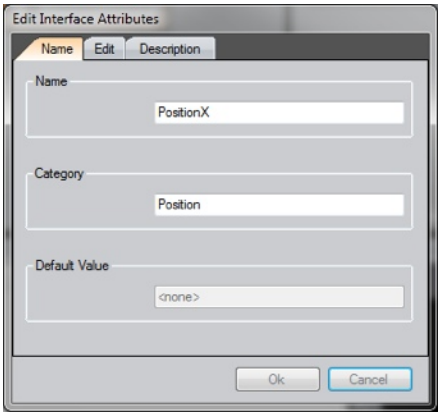
This button allows you to transform a container into an interface container and vice versa. Interface containers look different than normal containers in the Hierarchy and Content Editors:



A transformation into an interface container will create the interface based on all currently exposed properties. Interface properties can be recognized in the Property Editor by their bold surroundings:



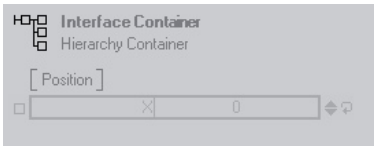
The interface attributes are created on the basis of the previously exposed properties with the difference that the “inherit” option does not exist anymore. Interface attributes must be strict and cannot rely on inherited values. To edit these attributes you can either use the context menu of the Properties Editor and select “Edit Interface …” or LMB double click on the property’s name.

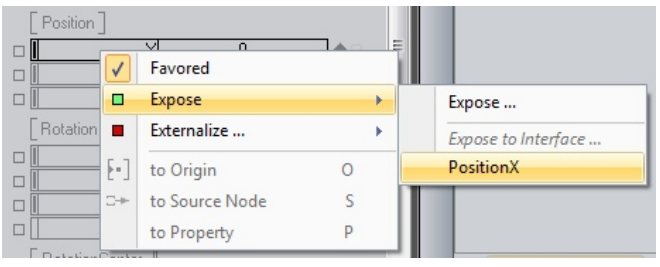


An interface container can be used like any other “normal” container. You can change its properties, bind them to other nodes or create an animation controlling them.

The major difference of interface containers is that if you unexpose a property inside these containers, the interface properties remain existent! You also could delete all nodes inside this container – all properties and their bindings will remain – but in a “disconnected” manner:

Disconnected interface properties can be recognized as being displayed transparently in the Property Editor. Disconnected properties are fully functional. They can be changed or bound to other nodes (or animated). They may lose special information, such as the type of text to be edited or the type of file which has to be opened in the File Open dialog of Ventuz. These additional attributes are restored if you re-connect an interface.





An interface container has all its properties as interface properties. That means if you expose further properties they will also appear as interface properties!

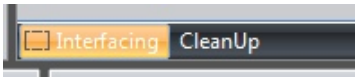
It is highly recommended to only use unique names for all interface properties, methods and events, because the "Interface-Match-Algorithm" could get confused when re-connecting interfaces (see below)

If you want to expose properties to an existing and disconnected interface property you can use the context menu of the Property Editor or simply click on the expose button ■ to the property which will "provide" the functionality. All interface properties "matching" the property to be exposed will be displayed in that menu.

(see below)

Remove Disconnected Interface Properties

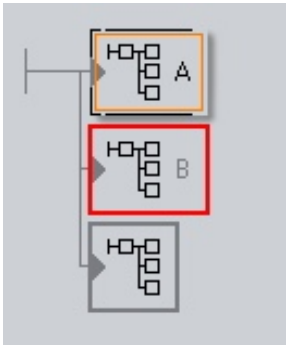
If you decide that some interface properties are not needed and you want to remove them from you container, follow these steps:

- Unexpose or delete the provider of these interface properties to bring the interface properties into the disconnected state
- Double click on the disconnected interface property. The "Edit Interface Attributes" dialog will open – press the "Exclude" button.
- Another optional way is to click the "CleanUp" button at the Property Editor while the interface container is selected. This will delete all disconnected interface properties!
- 

How to exchange the "Provider" of an interface container?

You treat an interface container as if it were a normal container – you can copy it, bind it or store it in your repositories. There is, however, one new bit of functionality which is a key feature of interfaces:

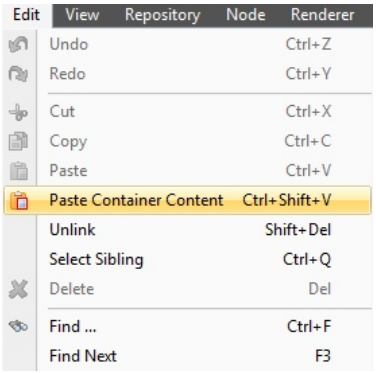
If you Drag&Drop an interface container, the Hierarchy and the Content Editor will highlight all other interface containers in your current view. If you drag over the center of such a container you won't see the common red arrows. Instead you will see the new red frame surrounding the "target container". If you now drop the content you will exchange the entire content of the target container for the contents of the container you dropped! What actually happens behind the scenes is that the content of the target gets deleted and replaced by the entire content of the source. After that the interface gets



reconnected using the interface matching algorithm (see below).

If you have to update multiple target containers with the same new content, you also can use the Drag&Stamp function: Start the drag operation normally and “stamp” the content inside the targets by clicking the RMB – keep the LMB pressed down during stamping!

Another way to exchange the content is using the clipboard functions:



- Select the source container and press Ctrl-C for Copy or select the appropriate entry in the main edit menu.
- Select the target container(s) and press Ctrl-Shift-V to paste the content

The Drag and Drop Content functionality is also available if you drag items from the Repository into your scene. That makes collaborative work possible, because another seat could “prepare” the providers based on an interface negotiated before!

An automatic Repository-Reference-Update is planned for the future.

Interface Matching Algorithm

Exposed and Interface Properties match each other if they fulfill the following requirements:

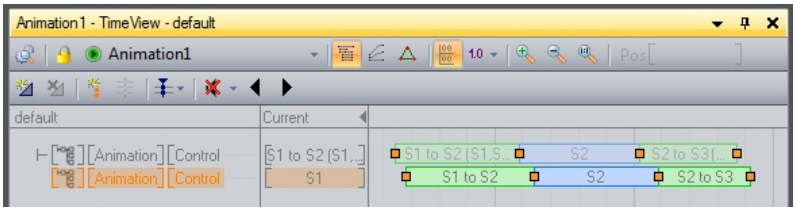
1. Property Type must be identical: Input Value, Output Value, Method, Event
    a. If the type is Input or Output Value, the SubType must be identical
    b. (float, integer, string, matrix, …)
2. Property Names must be identical.
3. The names are compared case sensitive while the category is ignored.
4. Please remember that property names starting with the same text as their category are displayed in the Property Editor without that leading portion: eg. “PositionX” in category “Position” is displayed as “X”

If the “provider” provides more interface properties than the target expects, the target gets extended by these properties.

If the “provider” provides fewer interface properties than the target expects, that target leaves the “missed” properties in disconnected state. They could be re-connected by inserting another provider later.

If the interface has interface properties of the same Type (and SubType) and same Name, the matching algorithm will be undefined. In that case, there is no guarantee how and in which priority those unidentical interfaces will be matched. So, please avoid using identical names for different interface properties!

How are disconnected interface properties handled by other bound nodes?

- Simple properties like float, integer, Boolean, enumeration, matrix simply keep their last transferred value. Other nodes communicate with these properties like normal, but without actual effect, because the interface container doesn't know what to do with it as long as no provider "connects" to them.
- Reference properties like Texture, ValueSwitch inputs, Image, Excel Workbook or any other resource are cleared when they are disconnected outputs. Disconnected inputs retrieve the reference but do not forward it to the provider, because there is none.
- Complex properties like Tester (MouseOver/HitTest), Animation Control, Arrange Control, Audio and Movie Control, Simple or TimeLine Control have a special behavior if they are "talking" with disconnected interface properties. While they are disconnected they store their reference symbolic (as text). An embedded animation channel, for example, stores all state and connection names in the channel and marks the channel as "disconnected". If the connection comes back, it tries to "recover" all references based on the symbolic name. Only if the recovery is 100% possible will the channel get "connected" again. A disconnected channel is drawn transparently in the Animation Editor. The State and Connection context menu on the Key Frames is not shown, because there is no sub-animation connected from which to retrieve this information.
- 

- Audio and Movie Control properties are treated the same way. The controlling Animation will recover the channel after re-connection. Media length information of the previously connected media cannot be displayed in disconnected state.
- All nodes (or animation channels) operate normally – even if they are disconnected!

Interface on Scene Ports (Professional and Broadcast Edition only)

Scene Port nodes (or Slide Ports) are always able to have an interface, so they are displayed always in the interface style.

Port interfaces have only one major difference to container interfaces: not all exposed properties of a scene are automatically part of the interface, rather they have to be defined manually!

Scenes can expose properties at the highest container level (top-level) in order to expose their properties to ports loading these scenes. You can see all top-level exposed properties by either selecting the scene in the Stage Editor or clicking in the free area of the Hierarchy Editor on the top level.

If a scene is assigned to a port you can create a port interface by selecting the corresponding port node. You will see all port properties plus all exposed properties of the current assigned/loaded scene of that port. If such properties are not connected to an interface yet, you see those properties merged with all other port properties but without the option to expose/externalize (the little expose button is not available ▫). You can operate these merged properties, but you are unable to bind them, as they are not really part of the enclosing scene. But you can create/define a port interface based on such “virtual” properties by simply LMB double clicking on their names. The “Edit Interface Attributes” dialog will be shown where you also can “Exclude” (remove) the port interface property again.

If the port gets re-assigned with another scene, the normal match algorithm takes place again!

Scene Port Interfaces cannot be dragged&dropped like the interface containers, because the actual “provider” is an entire scene. Use the Stage Editor to exchange providers (scenes) or create your own “switching-logic” by binding the Port’s “Scene” property.

Scene Port Interfaces are very advanced and used to modularize very large projects. Sometimes the view of the port properties in the merged manner can confuse the operator easily - use the port interfaces carefully.

Scenes are always disconnected and reconnected from their port interfaces during save or copy/paste operations. This is necessary to release all referenced nodes from other scenes while saving the content to disk. After the save (copy) process the interfaces get reconnected. Usually you won’t realize the disconnect/reconnect process, but it is important to know that it happens.

R5.24 New Message Log Window with Navigation

The Messages windows can now navigate to the node which has emitted a message. If the magnifier icon is visible next to the message line you will be able to navigate to the node if the scene containing the node is opened in Designer. Simply LMB double click on the message line.