Article
Lexi Hayden · Jun 8, 2017 3m read

Using the OnGetNodeInfo callback of a Zen <dynaTree>

I needed to use the OnGetNodeInfo callback of a Zen <dynaTree>, because this seems to be the only way to control the style separately for different levels of the tree. This post describes two discoveries I made.

How a dynaTree builds a tree via the OnGetNodeInfo callback

When you use a dynaTree with the OnGetNodeInfo callback, the dynaTree creates a series of nodes and displays them top to bottom *in the order they were created.*

The OnGetNodeInfo callback is an iterator. To create the nodes, the dynaTree calls the OnGetNodeInfo callback, each time passing the variable **pHandle** to it. Initially, **pHandle** is null. In the callback, you set **pHandle** as needed, to tell the dynaTree what to do *in the next iteration*. Also in OnGetNodeInfo, you use the information available in **pHandle** to create a node (or not). If you specify HasChildren=1 for the node, then this node is displayed as a folder, and the next node (if and when it's created) is indented.

Last, the OnGetNodeInfo must return either 1 or 0. If OnGetNodeInfo returns 1, the dynaTree creates the node you just described (in this iteration) and iterates again with this callback. If OnGetNodeInfo returns 0, the dynaTree does not create the node, and then stops iterating *within this indentation level* -- instead it pops out to the next highest level (or quits entirely).

How to create a complex structure within the OnGetNodeInfo callback

I had successfully written my OnGetNodeInfo callback to include an inner loop in addition to the outer loop. Then I found I needed to add another level of looping. To do this, it is necessary to pass (via **pHandle**), from iteration to iteration, information about where you are within a given loop, and this is tricky, because you have to invent a way to make it work. Then I thought, rather than deal with an unfamiliar iteration system (this callback), why not create a process-private global that contains the information I want, and then just read parts of that global? So that's what I did. ^||viewnodes contained the nodes I wanted, in order. ^||viewnodes has one subscript, which indicated the node number. ^||viewnodes(n) was =$LISTBUILD of the data I needed to define the node. I used **pHandle** to tell the dynaTree which node number to create.

So rather than using a new and unfamiliar iterator, I used a familiar iterator to build a global, which I then could iterate over very easily in the OnGetNodeInfo callback.

Finally, I had a bug in that the last node within any loop wasn't getting created. That's when I realized that *if OnGetNodeInfo returns 0, dynaTree does not create the node you specified in* that *iteration*. I needed a way to tell the dynaTree to finish this iteration but quit the next one. So I settled on this strategy: **pHandle** had the form *nodenumber*_"^"_"noquit" or *nodenumber*_"^"_"quit". Within any given iteration, I incremented the *nodenumber* part of this string and set the other part to either "noquit" or "quit".

The first thing that my callback does is to take apart **pHandle** and find out if it says to quit or not. If the second part of the **pHandle** string is "quit", that means, "don't do this iteration" -- so quit with 0. Otherwise proceed to create a node and quit with 1.

#ZEN #Caché