How to insert a large number of items into a treeview efficiently

devblogs.microsoft.com/oldnewthing/20111125-00

November 25, 2011

Raymond Chen

Just a quick tip today.

If you need to insert a large number of items into a treeview, like tens of thousands, then it's much more efficient to insert them "backwards". (I'm ignoring for now the usability question of having a treeview that large in the first place.) In other words, instead of

```
for (i = 0; i < array.Count(); i++) {
 TVINSERTSTRUCT tvis;
 tvis.hParent = hParentNode;
 tvis.hInsertAfter = TVIF_LAST;
 tvis.item.mask = TVIF_TEXT;
 item.item.pszText = array[i].Text();
 TreeView_InsertItem(hwndTreeView, &tvis);
}
```

do it this way:

```
for (i = array.Count() - 1; i >= 0; i--) {
 TVINSERTSTRUCT tvis;
 tvis.hParent = hParentNode;
 tvis.hInsertAfter = TVIF_FIRST;
 tvis.item.mask = TVIF_TEXT;
 item.item.pszText = array[i].Text();
 TreeView_InsertItem(hwndTreeView, &tvis);
}
```

Why is backwards-insertion faster?

It has to do with the annoying `hInsertAfter` parameter. To validate that the `hInsertAfter` parameter is valid, the treeview needs to verify that the `hInsertAfter` is a valid child of the `hParent`, and this is done by walking the parent's children looking for a match. The sooner you find the match, the faster the validation completes. (And if you pass `TVI_LAST`, then the treeview needs to walk to the end of the child list.)

You'd think that you could verify the parent/child relationship by just doing a `TreeView_GetParent(hInsertAfter)`, but that turns out not to be strict enough, because `hInsertAfter` might itself not be a valid parameter. If `hInsertAfter` is a bogus value, then you may crash when you try to read its Parent property. That's if you're lucky. If you're not lucky, then the random memory that `hInsertAfter` points to might look just enough like a valid `HTREEITEM` that you end up inserting the new node after a completely bogus node, and now the treeview has become corrupted.

Sure, you got the same problem if you passed a garbage `HTREEITEM` to `TreeView_GetParent`, but in that case, it's just garbage in garbage out. Nothing gets corrupted; the application just gets a garbage result. But in the case of `TreeView_InsertItem`, the treeview is going to update its internal data structures based on the garbage you passed in, and that means that the treeview winds up corrupted.

To ensure that the value passed in is valid, the treeview checks it against the list of valid values for `hInsertAfter`. And therefore, you get better performance if the valid value you pass is the one that it checks first.

(As it happens, a *lot* of programs pass garbage for `hInsertAfter`, so this defensive validation step is absolutely necessary.)

You might say that the treeview could have a one-off optimization for the special `TVI_LAST` value by remembering the last child so it can be located quickly. The question is whether the complexity of adding that optimization is worth the cost: Any tree rearranging function would have to update the cached value, and if you miss a spot, then the treeview ends up corrupted. That's a pretty high-risk optimization you're suggesting there.

And think about it this way: You're worrying about a tree where a single node has tens of thousands of children, something which (I am no longer ignoring) is a horrible user interface. That's like a list box with tens of thousands of items, or a dialog box with tens of thousands of checkboxes. You'll probably want to consider a better way of presenting the information than in a tree view that goes on for hundreds of screens. The treeview isn't optimized for this case because you don't optimize for the case where somebody is mis-using your system.

Raymond Chen

Follow