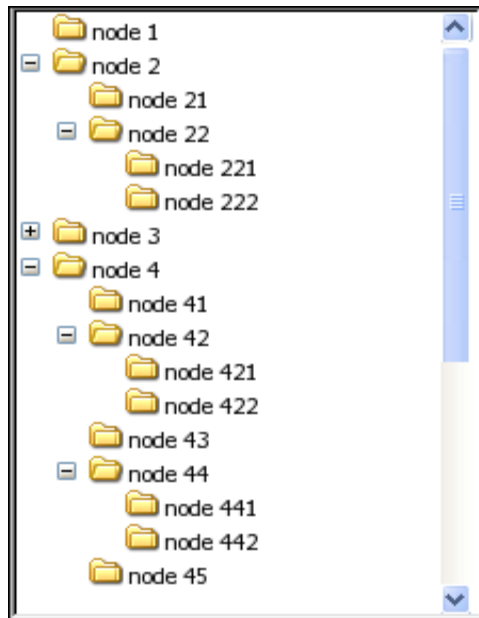


pure4glTv.w

A Treeview Smart Object in pure 4GL ABL
by Sébastien Lacroix, January 2005,
Revision on November 2007



Written with OpenOffice.org

Table of Content

0. New on November 2007.....	3
1 Why doing a treeview object in pure 4GL?.....	6
2 The specification was rather short:.....	6
3 Achieved Features:.....	6
3.0 How to use it? (a first drop of pure4glTv on an empty smart window).....	6

3.1	Look and feel like MS Windows XP File Explorer	9
3.2	One central tvNodeEvent event procedure	9
3.2.1	Only one single event is published from pure4gltv.w, with two parameters	9
3.2.2	The complete list of events is	9
3.3	A large set of APIs to control the treeview or interact with it	10
3.4	A few properties	11
3.4.1	wineMode (character)	11
3.4.2	windowsSkin (character)	12
3.4.3	picCacheCoef (decimal)	12
3.4.4	labCacheCoef (decimal)	12
3.4.5	tvIterationHeight (integer)	12
3.4.6	treeStyle (integer)	12
3.4.7	FocSelNodeBgColor and FocSelNodeFgColor (integers)	12
3.4.8	UnfSelNodeBgColor and UnfSelNodeFgColor (integers)	12
3.4.9	tvnodeDefaultFont (integer)	13
3.4.10	resizeVertical (logical)	13
3.4.11	resizeHorizontal (logical)	13
3.4.12	dragSource (character)	13
3.4.13	autoSort (logical)	13
4	Quick demo with the smartWindow testSmart4glTv.w	14
4.0	Testing navigation, resizing, and the events fired by pure4glTv	14
4.1	How is the treeview populated?	15
4.1.1	Simple static examples	15
4.1.2	A more practical case with nodes added on the fly	15
4.1.3	Database nodes added on the fly	15
4.2	Testing the drag and drop feature	16
4.3	Testing popup menus	20
4.4	Testing few APIs (with the colored widgets on the right)	21
5	A few Performance tests	21
5.0	Results on Linux / Wine (with a 2GHz Processor)	22
5.1	Results on Native MS Windows (with a 3.2GHz Processor)	23
6	API documentation	26
6.0	addNode	26
6.1	buildAndOpenPopupMenu	27
6.2	collapseNode()	28
6.3	deleteNode	28
6.4	deselectCurrentNode()	28
6.5	disableObject (new on July 2006)	28
6.6	enableObject (new on July 2006)	28
6.7	expandBranch (new on July 2006)	29
6.8	expandNode()	29
6.9	emptyTree	29
6.10	findNextNodeToShow	29
6.11	findPrevtNodeToShow	30
6.12	fMainKeyEvent	30
6.13	getNodeDetails	30

6.14	getNodeId().....	31
6.15	getNodeKey().....	31
6.16	getNodeLocatedAtXY().....	31
6.17	getNodeParentKey().....	31
6.18	getSelectedNodeKey() (new on July 2006).....	31
6.19	goToNode().....	31
6.20	lastVisibleNode().....	32
6.21	loadDemoTv.....	32
6.22	moveNode.....	32
6.23	nodeAtVirtualIter().....	33
6.24	nodeInCollapsedBranch().....	33
6.25	picFileName().....	33
6.26	resizeObject.....	33
6.27	selectNode().....	34
6.28	sortChildren.....	34
6.29	swapNode.....	34
6.30	topNode().....	35
6.31	updateNode.....	35
6.32	updateNodeWidth.....	35
6.33	tvRefresh().....	36
6.34	tvScroll().....	36
6.35	widgetsAt.....	36
7	Many Thanks to.....	36

0. New on November 2007

</Extract from comments in the definition block of pure4gltv.w>

Simon L Prinsloo, Dries Feys and slacroix 28-NOV-2007

Add a double-click event on the nodes in the bottom of the tree, provided that the "AddOnExpand" is not set for the node.

New node options for the addNode API:

fgColor=
bgColor=
font=
tooltip=

See demo usage of new these new options in the choose trigger of the "Small TV" button of the provided testSmart4glTv.w sample.

slacroix 04-AUG-2007

Fixed regression introduced with the search on keyStroke for the '-', '+' and '*' key events
Indeed, they were caught by the focusCatcher combo-box to perform search on keyStroke
instead of letting the fMainKeyEvent procedure doing the initial collapse,

expand or expandAll functionalities

slacroix 25-JUL-2007

1) Added support of the Aero skin for winVista

2) Review to remove errors returned by functions for compliance with more consistent error handling introduced in 10.1C

Note: In the past, I had a few FUNCTION capable of returning an ERROR: CollapseNode(), ExpandNode() and NodeInCollapsedBranch()

Actually, the point was more to show my design intention for cases that should not occur. The AVM could not raise an error in the caller at this time (it will in 10.1C), so it was not a problem to just return NO or unknow instead

slacroix 19-JUN-2007

New option in addNode of pure4glTv so superceed the autoSort TV option for one single node, and made abhack use it for the parameters. See header comment of addNode() for more info

slacroix 16-JUN-2007

1) New ApplyEntry strong override to really apply entry to the treeview by using the focusCatcher combo.

2) New search by keyStroke feature. When the focus is in the treeview (actually in the hidden focusCatcher static combo), pure4gltv now maintains a string composed by the last keystrokes with up to 1500ms between each strike (above this time, the string is reset to blank), then searches the next brother node (same parent as current node) which label begins with the mentioned string. The search uses an index against parent + label that has always been present, so this feature is very fast and cool. When we cannot find a next brother based on the searched label, then pure4gltv tries to find a first brother node. Of course, when the search is successful, then the found brother node is selected and brought into the viewPort if necessary with the selectNode API

3) Added protection against unsupported skins like aero now for WinVista (until someone sends me the necessary pictures). When I find out the theme is not supported in the tvSkins directory, pure4gltv goes into classic mode and gives some details in the tooltip of the EMPTY square on the right bottom corner.

slacroix 09-JUN-2007

1) implemented a new nodeDisplay event a bit similar to a Browse ROW-DISPLAY event:

/* 08-APR-2007 sla: new nodeDisplay hook to give the ability to refine a node just before it gets displayed it's up to the container to subscribe to this event. It is a bit like a ROW-DISPLAY trigger in a browse

This way, one can refine node labels just before they get displayed. One nice example is a new `prospy++` that displays a huge trace in a treeview with time information. This time info can be either an absolute time or a relative time, or a node cost or branch cost. In the past, without this hook, I had to reupdate the labels of the all treview when the end user was choosing a different display mode, which was very slow when the treeview was containing half a million of nodes.... Now, it is just **instant** since I refresh I prepare the node label just when it is about to be displayed ;) */

2) Provided additional pictures to be used in a new `prospy++` and in `ABHack`

slacroix 09-JUN-2007

1) implemented a new `nodeDisplay` event a bit similar to a `Browse ROW-DISPLAY` event:

```
/* 08-APR-2007 sla: new nodeDisplay hook to give the ability to refine a node just before it gets displayed it's up to the container to subscribe to this event. It is a bit like a ROW-DISPLAY trigger in a browse
```

This way, one can refine node labels just before they get displayed. One nice example is a new `prospy++` that displays a huge trace in a treeview with time information. This time info can be either an absolute time or a relative time, or a node cost or branch cost. In the past, without this hook, I had to reupdate the labels of the all treview when the end user was choosing a different display mode, which was very slow when the treeview was containing half a million of nodes.... Now, it is just **instant** since I refresh I prepare the node label just when it is about to be displayed ;) */

2) Provided additional pictures to be used in a new `prospy++` and in `ABHack`

slacroix & Nicolas Andricq MAR-2006

1) Support of Royale skin. Restructuration to support multiple skins

2) Review of the vertical scrollbar to avoid a hole of two pixels on the right.

3) new `getSelectedNodeKey()` API

slacroix Later in MAR-2006

1) New `expandBranch` API that is fired on '*' key trigger

2) New `enableObject` and `disableObject` API's

3) New `expandAll` functionality on '*' key

4) Made a new version of `Prospy` that uses `pure4gl tv` for greater performance

</Extract from comments in the definition block of `pure4gltv.w`>

1 Why doing a treeview object in pure 4GL?

- This was a great challenge.
- It proves that the 4GL is not only a great language to handle Database Business Logic, but is also very powerful at handling the logic of sophisticated GUI Objects, especially with the use of the powerful 4GL temp-tables. Indeed, in the past, the point of the UI was to manage data in a database. Nowadays, it is almost the other way round with sophisticated UI objects that almost require a database system to manage them. Simply, I was wondering how one can achieve a treeview without using database objects like temp-tables.
- This object provides an alternate solution to using a treeview and an image list OCX. This is a way to avoid potential deployment and license problems.
- Finally, this object works fine on Linux with Wine. Pure 4GL based GUI objects like SmartFolders SmartSelectors or SmartCalendars work fine on this configuration, on the other hand OCX's do not work (or at least make problems). This was preventing the use of a treeview object, which is rather important. Therefore pure4glTv helps to open the door to Linux/Wine. Note that this project has been mainly developed at home on Linux with Wine.

2 The specification was rather short:

- Achieve a Treeview object in pure 4GL, or with a bit of external programming techniques (few calls in few DLL's) so it can work on both MS Windows and Linux with Wine (without any native Microsoft DLL). The idea is to provide an alternate solution to using an ActiveX.
- This treeview has to look and behave like the treeview of the MS Windows XP File Explorer.

3 Achieved Features:

The treeview is a SmartObject™ so it can be used as an object in any smart container (so in a smart window or a smart frame). Please note this is not an attempt to push people to adopt the complete ADM2 framework. One just needs to use smart windows or smart frames so it requires a very little knowledge of ADM2. In other words, SDOs, smart viewers and smart browser are not required. I did not even have the time to prepare a sample code with a few SmartDataObjects and SmartDataViewers, and the example with nodes to display the content of a few database tables directly refers to the database like in pure client server.

3.0 How to use it? (a first drop of pure4glTv on an empty smart window)

With the AppBuilder window, create a new smart window as shown in Illustration 1 (It is actually faster to right click on *New* button then choose the *SmartWindow* popup menu item). Then on the object palette, choose the *SmartObject* button (see Illustration 2), choose the pure4gltv.w object and drop it on the new empty smart window.

You should have a window with a very small treeview looking like Illustration 3.



Illustration 1: The simple Smart Object button on the palette

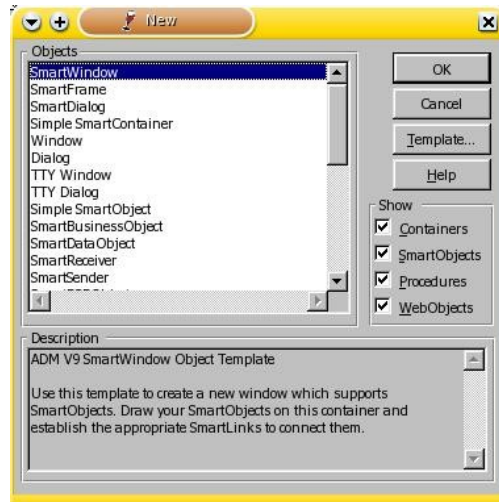


Illustration 2: AppBuilder -> New -> choose 'Smart Window'

Then resize the new instance of the pure4gltv to make it a bit bigger so you can see a cute little demo treeview as shown in Illustration 4.

Next, right click on it, choose the SmartLinks menu item then choose the Add button. Select *h_pure4gltv* as Source, *THIS-PROCEDURE* as Target (this one is the smart window) and *New...* as Link type. When you get the New Link Type dialog then type *tvNodeeEvent* to obtain something like Illustration 5.

With that, the ADM2 will make *THIS-PROCEDURE* (the window) subscribe to *tvNodeeEvent*. At this point, you have a window that is ready to handle the *pure4glTv* object, like the given sample *testSmart4glTv.w*, that is described a bit further in this document.

If one does not want to use smart links, then he can just type the following code in *initializeObject* of the smart window (after the *RUN SUPER*. Should be fine):

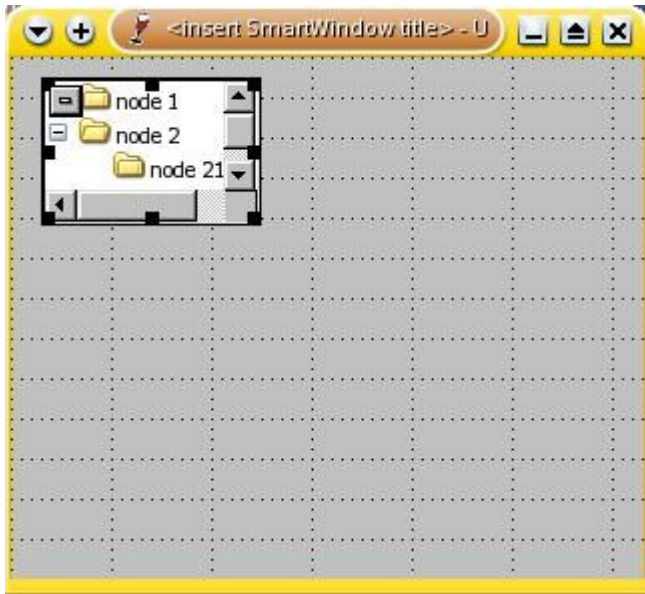


Illustration 3: Pure4glTv was just dropped on a new Smart Window

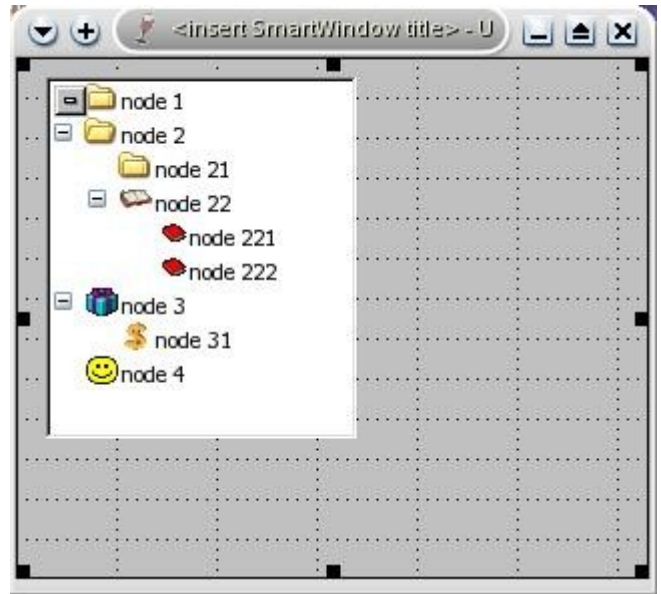


Illustration 4: Pure4glTv has been resized. It shows a cute demo treeview

SUBSCRIBE TO "tvNodeEvent" IN h_pure4gltv.

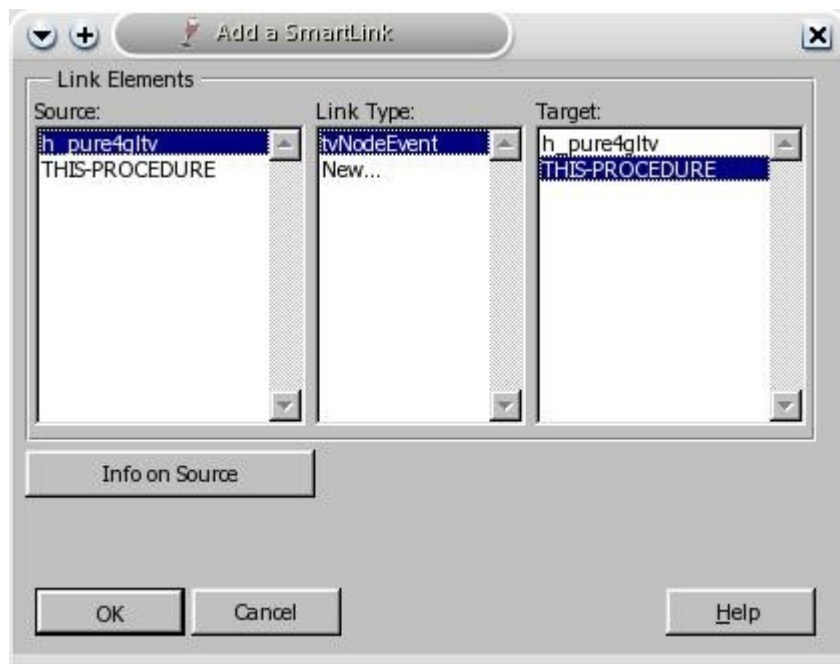


Illustration 5: The Add a smart Link dialog. An efficient way to make the window SUBSCRIBE TO tvNodeEvent IN h_pure4glTv

One advantage of using smart links is that there is no need to hard code the name of the treeview instance (here *h_pure4gltv*) as the AppBuilder will take care of it. There are other advantages, but the point of this document is not to be an ADM2 course.

3.1 Look and feel like MS Windows XP File Explorer :

3.2.1. Navigation keys, selection, collapse/expand, click, double click, + and -

3.2.2. Supported Windows skins are *XPStyle* and *Classic*. This might be extended in the future, by adding the popular *Silver*.

3.2.3. Scrollbars:

- Native for horizontal Scrolling
- Fully emulated and controlled for Vertical Scrolling. Again same behavior for all manipulations like grab button, click above/bellow, and the flat down and up buttons.

3.2.4. Scroll by 3 iterations with the mouse wheel.

Note that this feature does not work on Linux KDE. Sadly, it seems to be due to a smarter way to handle the mouse wheel with KDE: if a scrollable widget is bellow the mouse pointer, then a mouse wheel action immediately gives the focus to its window (without moving it to top, depending on some KDE setting), then it scrolls the scrollable widget. This is very handy to read multiple documents in multiple windows at the same time.

Now, I am relying on a hidden combo-box that keeps the focus until we leave the treeview. This combo catches the mouse wheel scrolls by selecting a next or previous item and firing the `VALUE-CHANGED` event. The trouble is KDE finds out that this combo is not visible at the top, so it does not fire the action described above. Changing some options in the setting of KDE might solve this problem, but the default handling of the mouse wheel of KDE is so nice for other applications that one may not want to change it (at least me).

3.2 One central tvNodeEvent event procedure

3.2.1 Only one single event is published from pure4gltv.w, with two parameters:

```
PUBLISH 'tvNodeEvent' (INPUT pcEventName, INPUT pcNodeKey)
```

By having one single event procedure then it is easy to link the Treeview to any object with a custom link (far easier than handling a list of Supported Target events in some custom include files)

One may also just do `SUBSCRIBE TO 'tvNodeEvent' IN h_pure4gltv`. Note that there can be multiple subscribed listeners, however few events rely on a `RETURN-VALUE` (like `dragBegin` and `rightClick`)

3.2.2 The complete list of events is:

- `select`
- `deselect`
- `expand`
- `collapse`
- `rightClick` Can build popup menus, see the `buildAndOpenPopupMenu` API
- `addOnExpand` Fired when a node with option `addOnExpand` is expanded

- **DragBegin**
- **DropEnd** See the code of dragNode for details about parameters
- **Custom events for the CHOOSE of popup menu-items.**

3.3 A large set of APIs to control the treeview or interact with it:

All the internal procedures and function that are not supposed to be called from the outside are defined with the PRIVATE option.

Bellow is the complete list of APIs that are available to the developer. More details about each API are given in section 6) *API documentation*.

Note that this list does not include the get<Property> and set<Property> functions. These properties are described further.

Note also that the APIs marked with two stars (**) were initially designed for internal use but I made them public afterwards because they might be interesting.

- Addnode
- buildAndOpenPopupMenu
- collapseNode()
- deleteNode
- deselectCurrentNode()
- expandNode()
- emptyTree
- findNextNodeToShow
- findPrevtNodeToShow
- fMainKeyEvent **
- getNodeDetails
- getIdNode()
- getNodeKey()
- getNodeLocatedAtXY()
- getNodeParentKey()
- goToNode()
- lastVisibleNode()
- LoadDemoTv
- moveNode
- nodeAtVirtualIter()
- nodeInCollapsedBranch()
- picFileName()
- resizeObject
- selectNode()
- sortChildren

- swapNode
- topNode()
- updateNode
- updateNodeWidth **
- tvRefresh()
- tvScroll()
- widgetsAt **

3.4 A few properties

Illustration 6 shows the instance property sheet of pure4glTv. For non ADM2 users: you obtain it by doing a right-click on a pure4glTv instance and choosing the *Instance properties* menu item.

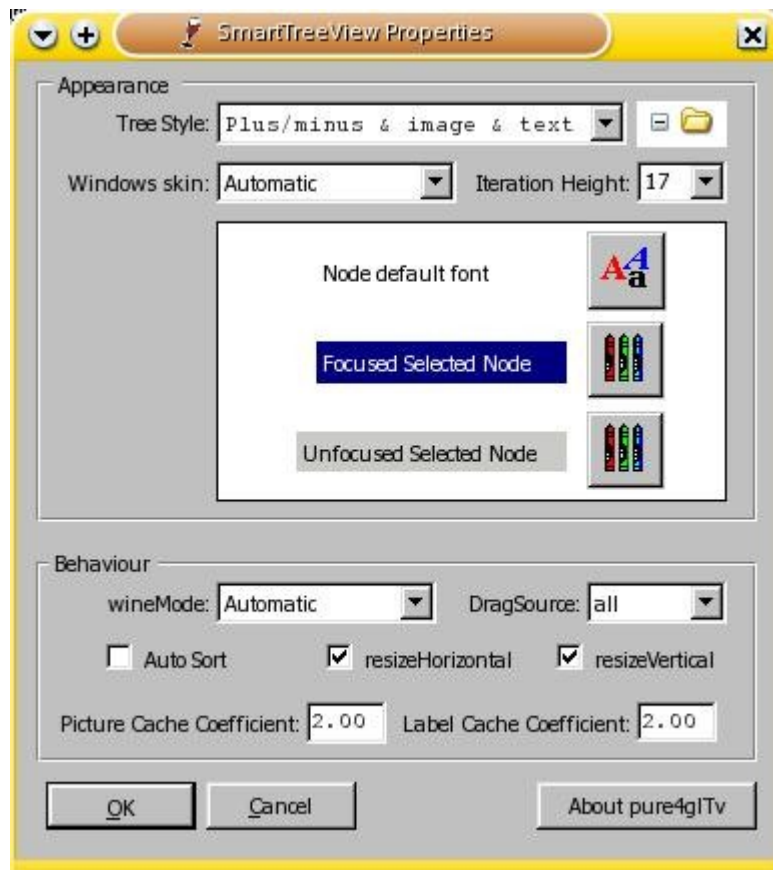


Illustration 6: Instance Properties sheet of pure4glTv

Bellow is a description of each property:

3.4.1 wineMode (character)

This property should remain set to *Automatic*. Other values are available mainly for testing. The point of this property is only to use the most appropriate optimization procedure for the rendering. There are

2 different optimization procedures because the cost of changing attributes for some GUI widgets is not the same for Wine and for MS Windows. See the `optimizeDisplay.xls` spreadsheet and the code of `optimizeTviterMSWin` and `optimizeTviterWine` for more details.

Internally, the object uses a variable called `glWineMode` that defaults to `NO`, so the property can remain set to *Automatic*. When *Automatic*, then `glWineMode` is switched to `YES` if we can find a key with name *RunningOnWine* set to *yes* in the *Startup* section of the environment (registry or ini file).

I decided to rely on a key in the Progress environment because I am not aware of any easy way to guess whether we are running on native MS Windows or on Linux with Wine. Apparently, Wine does not provide any possibility to escape from it (i.e. it does not allow the call of a native Linux command through it), which is perhaps not that bad for security...

3.4.2 windowsSkin (character)

This property should remain set to *Automatic*. Other values are there for testing the XP scroll bar when developing on Linux, or the classic scrollbar when developing on Windows XP with the fancy Fisher Price theme.

3.4.3 picCacheCoef (decimal)

Default is 1. Greater value can improve rendering performance (especially on Wine) by enabling a kind of over-caching with a higher number of rendering images than the number of iterations. This way, some widgets are just kept ready and hidden and can quickly be reused by switching their `VISIBLE` attribute (think of expand and collapse operations).

3.4.4 labCacheCoef (decimal)

Same as `picCacheCoef` for the label dynamic texts.

3.4.5 tvIterationHeight (integer)

Default is 17. A larger value will result in higher iterations with higher node pictures and labels. Note that currently, the width of pictures is not variable, but it would not be hard to achieve a new property.

3.4.6 treeStyle (integer)

Can have one of the following values: *Image & text*, *Plus/minus & text* and *Plus/minus & image & text*. The value is an integer code that is compatible with the codes used by the usual MS OCX.

3.4.7 FocSelNodeBgColor and FocSelNodeFgColor (integers)

To handle the color of the selected node when the treeview has the focus.

3.4.8 UnfSelNodeBgColor and UnfSelNodeFgColor (integers)

To handle the color of the selected node when the treeview has not the focus.

3.4.9 tvnodeDefaultFont (integer)

For now, the same font is used for all the nodes. We could rather easily handle it as a default font that could be overridden for some given nodes (as an option in the option list param of addNode). To achieve that, we have two possibilities:

a) The hard way: to handle the font info in the tvLab temp-table and perhaps handle that in the rendering optimization procedures, especially if we find out that changing the font dynamically cost a non negligible time compared to changing the X, Y VISIBLE or SCREEN-VALUE attributes of a dynamic TEXT widget.

b) State that changing fonts is rather rare so it would have a rather negligible impact on performance. So we could actually provide a hook to refine a label widget, like a refineLabelProcedure node option in the option list parameter of addNode (a RUN NO-ERROR in a super proc for example).

Anyway, I prefer to anticipate this future need and I therefore call this property tvnodeDefaultFont instead of tvnodeFont.

3.4.10 resizeVertical (logical)

It is up to the container to query this property in order to decide if it may resize pure4glTv vertically with the resizeObject API. Sending an unknown value for the height in resizeObject results in not changing it. See the trigger code of WINDOW-RESIZED of testpure4glTv.w for an example.

Note this property is a standard ADM2 property. In other words, it is not created only for pure4glTv.

3.4.11 resizeHorizontal (logical)

Same as for resizeVertical for the horizontal resize.

Note this property is a standard ADM2 property. In other words, it is not created only for pure4glTv.

3.4.12 dragSource (character)

Can be set to *none* or *some* or *all*. When set to *some* or *all* then the two node option *dragSource* and *noDragSource* are taken into account to allow a drag operation. See the source code of addNode for more details about these two node options.

Note also that another way to disable a drag operation is to make *tvNodeEvent dragBegin* return the keyword *cancelDrag*. See the comments in the source code of dragNode in pure4gtv.w for more details about the *dragBegin* and *dropEnd* events, or for any further information.

3.4.13 autoSort (logical)

Taken into account only when adding a node, so during the execution of the addNode API. Once a node has been added, it can be moved anywhere in the tree (although there are a few rules) with the moveNode or swapNode APIs. See the code of these two APIs for more information regarding the move of nodes.

4 Quick demo with the smartWindow testSmart4glTv.w

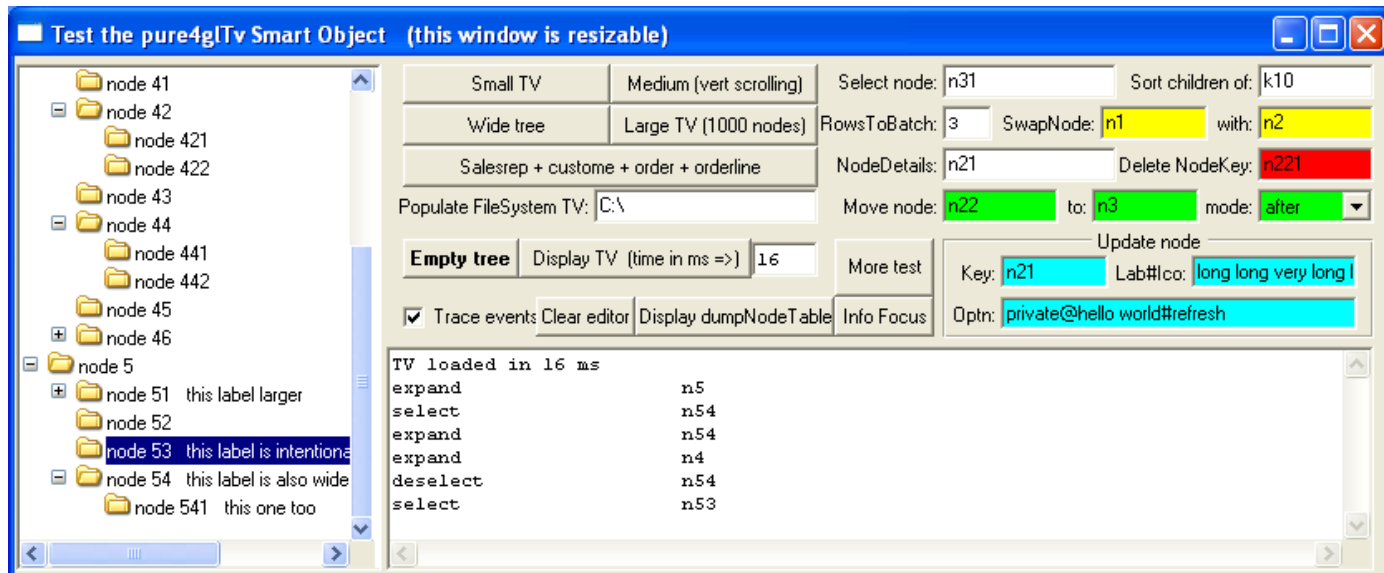


Illustration 7: testSmart4glTv.w, a single window to demo multiple uses of pure4glTv

4.0 Testing navigation, resizing, and the events fired by pure4glTv

Having a session connected to sports2000**, run testSmart4glTv.w. Choose the *Wide Tree* button to load a tree as shown in Illustration 7. Expand node n5, navigate in the Tree with the keyboard and see that you can go up and down, or expand or collapse. Try also the Page-up/Down or home and end keys, or even the plus and minus keys in the numeric key pad (only on a node that can be collapsed or expanded). Indeed, pure4glTv just behaves like the native treeview of the MS File Explorer.

** If you really do not want to be connected to a sports2000 demo database, then you might comment the code that refers to it. Actually, there is only one sample treeview that uses it (the *Salesrep + custome + order + orderline* button), but it is handled in a few procedures...

Next, testSmart4glTv.w is a resizable window so the height and width of the treeview is always adapted to fit the window. Try to resize it in various ways and have a look at the code in the WINDOW-RESIZED trigger to achieve the same kind of resizing.

See the events that are fired by pure4glTv in the editor widget when navigating in the tree. They consist in an event name and a node key. Notice that the tvnodeEvent procedure in the container is a kind of event broker. The advantage of doing that is that there is only one smart link called tvnodeevent to add between the container and pure4glTv (see *smart link* with a right click on a pure4glTv instance in the container) and there is no need to maintain a list of target and source events to subscribe to.

4.1 How is the treeview populated?

4.1.1 Simple static examples

See the code in the choose trigger of the *Medium (vert scrolling)* button and see how the addNode API is called. See also the code for the *Small TV* button and notice that the last parameter is actually a extendable list of options that is CHR(1) separated (see for node n3). For more info, see details about the addNode API.

4.1.2 A more practical case with nodes added on the fly

Clear the container again, and press the RETURN key in the *Populate FileSystem TV* fill-in, having 'C:\' in its screen-value to obtain a tree like the one shown in Illustration 8. Indeed, one could easily achieve a file explorer with pure 4GL... Look at the code of loadDirectory and see how short it is.

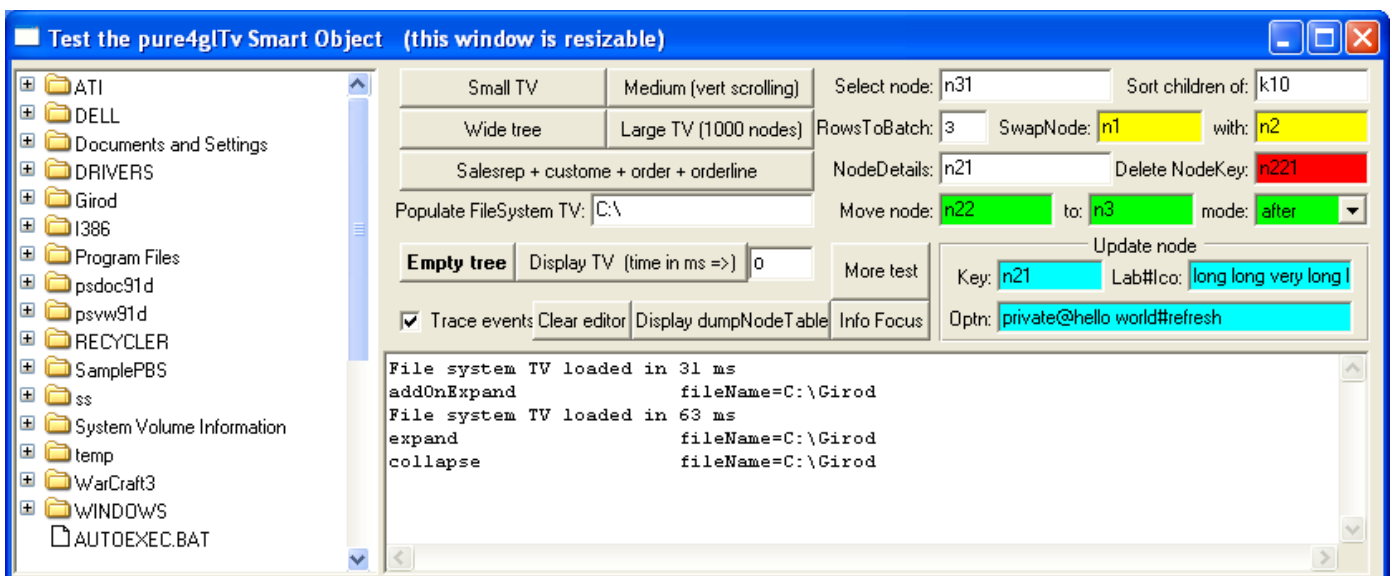


Illustration 8: A kind of File Explorer with nodes added on the fly by using the FILE-INFO system handle

Next, expand the node of some folders. Yes, the situation is handled with sub nodes added on the fly with the following code in tvNodeAddOnExpand of the container testSmart4glTv.w:

```
[...]
IF pcNodeKey BEGINS "fileName=" THEN DO:
    cFullPath = ENTRY(2,pcNodeKey,"=").
    RUN loadDirectory (pcNodeKey, cFullPath).
END.
```

4.1.3 Database nodes added on the fly

This example is certainly the one that will illustrate the biggest need with treeviews.

Clear the container again, and choose the *Salesrep + custome + order + orderline* button. Expand few salesrep, choose the *More...* node to get more customers, expand order and see the order lines. Indeed

the nodes are added on the fly. Note that only 3 nodes are added at a time before creating a 'More...' node. This is very small for test purposes but one can well change the value in the RowsToBatch fill-in to something more practical like 50. Note also that the concept of More node belongs to the container, not to pure4glTv itself, which just passes the event *select* to the container (see the tvNodeEvent procedure that calls tvNodeSelect). In other words, pure4glTv was designed to accept the addition of nodes on the fly when selecting a node without any special *More Node* option.

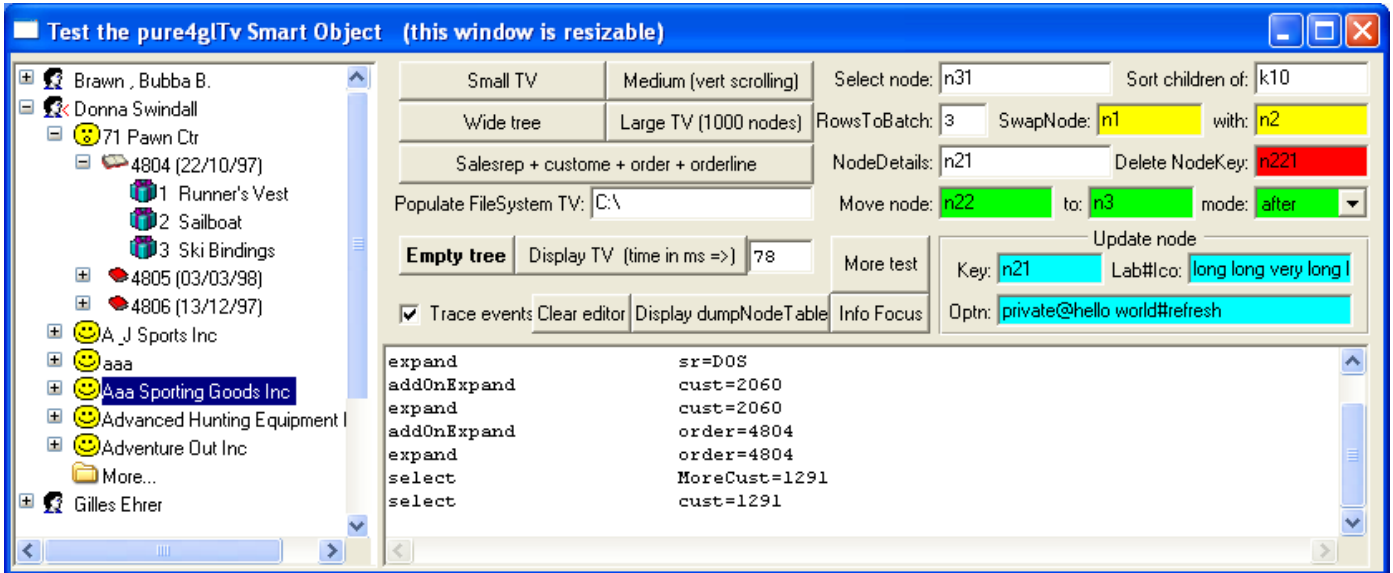


Illustration 9: Nodes added on the fly to display the content of 4 levels of parent child database tables

4.2 Testing the drag and drop feature

Load a treeview with the *Small TV* or *Medium* button. Drag *node 45* and move the mouse in the window. You see that a DragBegin event has been fired for the node with node key *n45* as shown in Illustration 10.

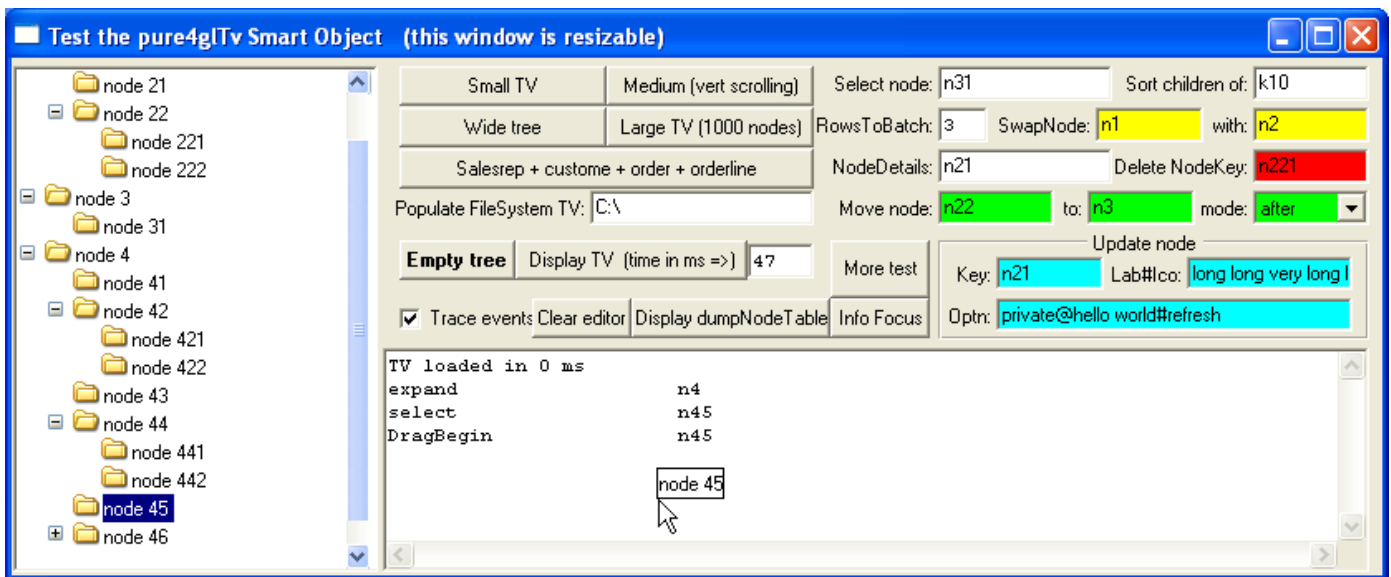


Illustration 10: Drag an drop within the container window (it is also possible to drop in another window)

Then Drop the *node 45* drag-box in the editor. You get a DropEnd event with a few information like the node key of the dropped node, the coordinates of the mouse pointer, and finally the widget handle of the drop target, as shown in Illustration 11.

You will also notice that the node label *node 45* has been automatically inserted in the editor.

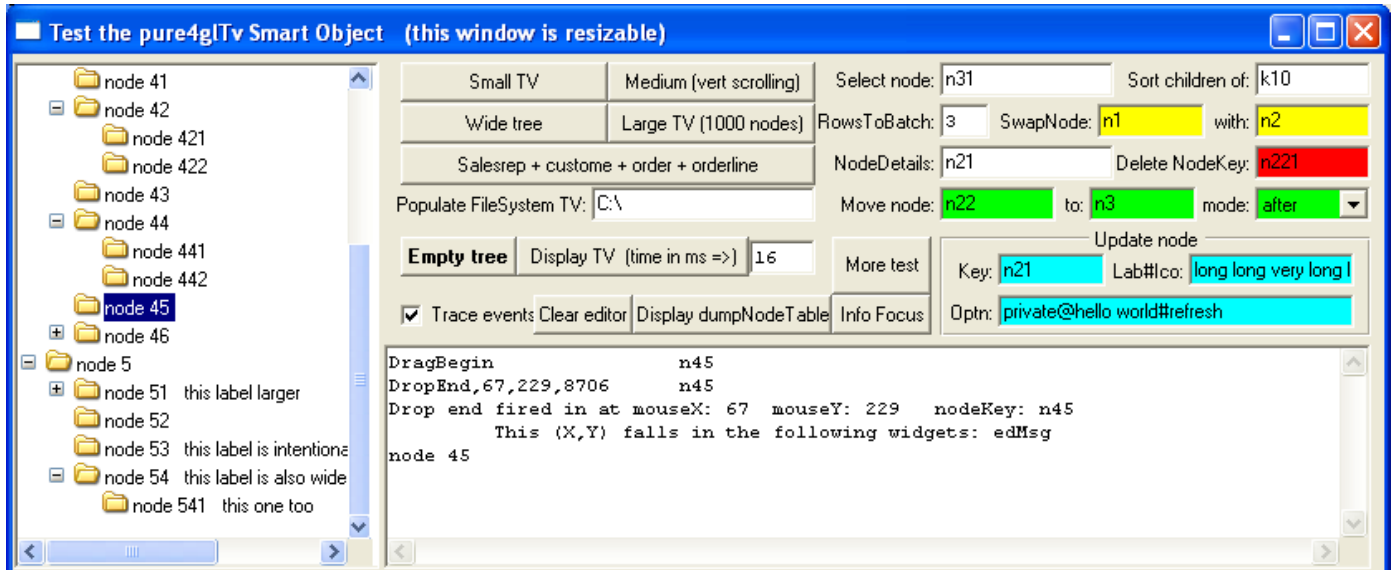


Illustration 11: The DropEnd event is actually completely handled by the container

Actually, the drop end process has been completely handled by the container smart window with the info supplied by pure4glTv, which is sensible enough to not achieve such a drop-node-insert-label by itself.

See the code of the internal procedure tvNodeDropEnd in testSmart4glTv.w. The following code finds out the drop target widget names (or screen-value):

```

/* work out the name of drop target widgets from the handles passed in pcEvent*/
DO iCount = 4 TO NUM-ENTRIES(pcEvent):
  hWidget = WIDGET-HANDLE(ENTRY(iCount,pcEvent)) NO-ERROR.
  cWidgets = cWidgets + " "
  + IF hWidget:NAME = ?
    THEN (IF CAN-QUERY(hWidget,"SCREEN-VALUE")
          THEN "SCREEN-VALUE=" + hWidget:SCREEN-VALUE
          ELSE "?")
    ELSE hWidget:NAME.
END.

```

And a rather generic code inserts the node label in the drop target widget:

```

/* at last insert the label of the dragged node into the drop target widget */
hWidget = ?.
DO iCount = 4 TO NUM-ENTRIES(pcEvent):
  hWidget = WIDGET-HANDLE(ENTRY(iCount,pcEvent)) NO-ERROR.
  IF NOT CAN-QUERY(hWidget, "SCREEN-VALUE") THEN NEXT.
  IF NOT hWidget:SENSITIVE THEN NEXT. /* otherwise, we give the ability[...]*/

  DEFINE VARIABLE hNodeBuffer AS HANDLE      NO-UNDO.
  RUN getNodeDetails IN h_pure4gltv
  (INPUT pcnodeKey /* CHARACTER */,
   OUTPUT hNodeBuffer /* HANDLE */).

  IF NOT VALID-HANDLE (hNodeBuffer) THEN LEAVE.

  CASE hWidget:TYPE:
    WHEN "EDITOR" THEN DO:
      hWidget:CURSOR-OFFSET = hWidget:LENGTH + 1.
      hWidget:INSERT-STRING(hNodeBuffer:BUFFER-FIELD("lab"):BUFFER-VALUE + "~n").
    END.
    WHEN "FILL-IN" THEN hWidget:SCREEN-VALUE = hWidget:SCREEN-VALUE +
      hNodeBuffer:BUFFER-FIELD("lab"):BUFFER-VALUE NO-ERROR.
    OTHERWISE hWidget:SCREEN-VALUE = hNodeBuffer:BUFFER-FIELD("lab"):BUFFER-VALUE NO-ERROR.
  END CASE. /* CASE hWidget:TYPE: */
  APPLY 'VALUE-CHANGED' TO hWidget. /*very important if we want the change of the SCREEN-
VALUE
                                     to result in the same as typing */

  DELETE OBJECT hNodeBuffer.

  LEAVE. /* one widget is enough ;) */
END. /* DO iCount = 4 TO NUM-ENTRIES(cWidgets): */

```

Then, choose the *Empty tree* and *Clear editor* buttons in order to get a clear test window again. Then choose the *Large TV (1000 nodes)* button.

Try to drag a node and drop it on another node. You will end up with a situation like Illustration 12.

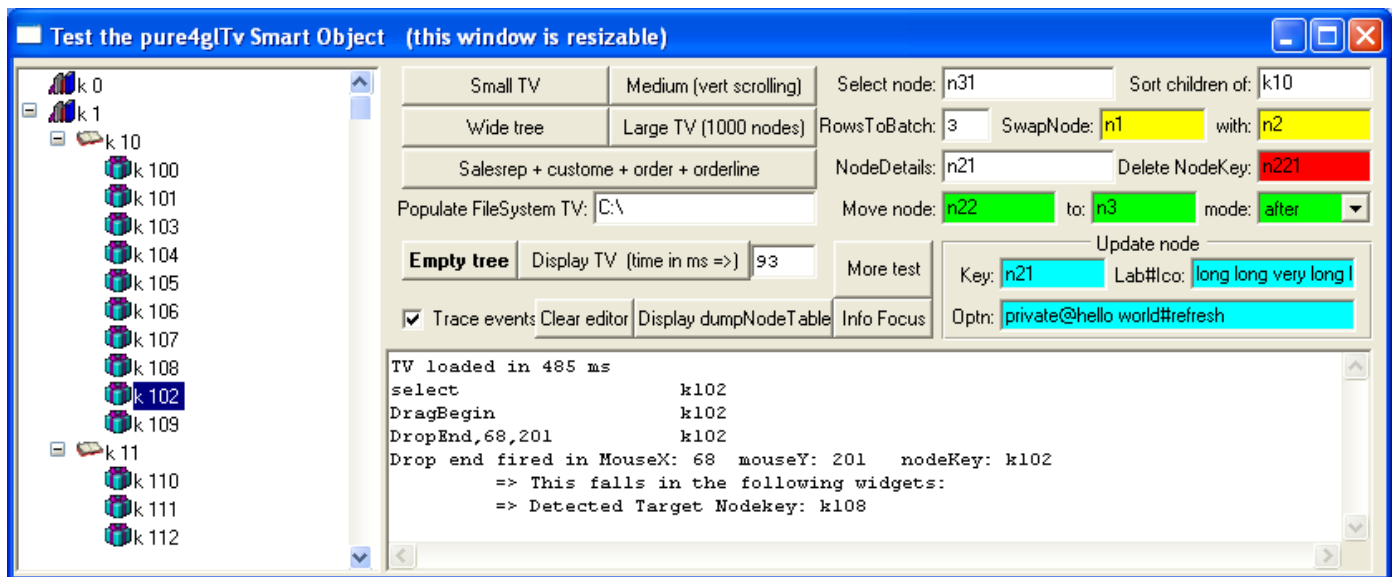


Illustration 12: One can use drag and drop on the treeview itself in order to move nodes in the tree

Indeed, the container window has asked the treeview to move the node *k102* after node *k108* with the following short piece of code:

```
/* Example with the large tree (1000 node) and drag-drop used to move
a node somewhere else (drop in the treeview itself) */
IF pnodeKey BEGINS "k" THEN DO:
    DEFINE VARIABLE targetKe AS CHARACTER NO-UNDO.
    targetKe = DYNAMIC-FUNCTION('getNodeLocatedAtXY' IN h_pure4glTv, mouseX, mouseY).

    [some tracing code here]
    IF targetKe <> "" AND targetKe <> pnodeKey THEN
        RUN moveNode IN h_pure4glTv (pnodeKey, targetKe, "after", "refresh") NO-ERROR.
    IF NOT ERROR-STATUS:ERROR THEN DYNAMIC-FUNCTION('selectNode' IN h_pure4glTv , pnodeKey).
    ELSE MESSAGE "This node cannot be moved here!"
        VIEW-AS ALERT-BOX INFO BUTTONS OK.
END.
```

A last demo with drag and drop: clear the window again, choose the *small TV* button, right-click on the *MoreTests* button and choose the menu-item *Run OtherDropTargetWin.w*. Move this *otherWindow* away. Drag the node *node 1* (drop on another window) and move the mouse pointer in the container window then in the *otherWindow*. Notice that a new set of dynamic widget is following the mouse pointer in the other window. How is this done? Again, from the container window, with the following code in *tvnodeEvent*:

```
WHEN "DragBegin" THEN DO:
    [...]
    /* drop target frame is in another window */
    IF pnodeKey = "n1" THEN DO:
        DEFINE VARIABLE hTargetFrame AS HANDLE NO-UNDO.
        /* that window has a SUBSCRIBE TO "getOtherWinTargetFrame" ANYWHERE */
        PUBLISH "getOtherWinTargetFrame" (OUTPUT hTargetFrame).
        IF VALID-HANDLE(hTargetFrame) THEN RETURN STRING(hTargetFrame).
    END.
```

Of course, here we rely on the node key (which code is up to the developer) to keep the example simple, but one may rather rely on a node property to query with the *getNodeDetails* API.

4.3 Testing popup menus

Clear the treeview and choose the *Salesrep + custome + order + orderline* button. Expand a salesrep and an order then see the order lines. Now right-click on an order node. You get a *Add order line* menu item in a popup menu as show in Illustration 13.

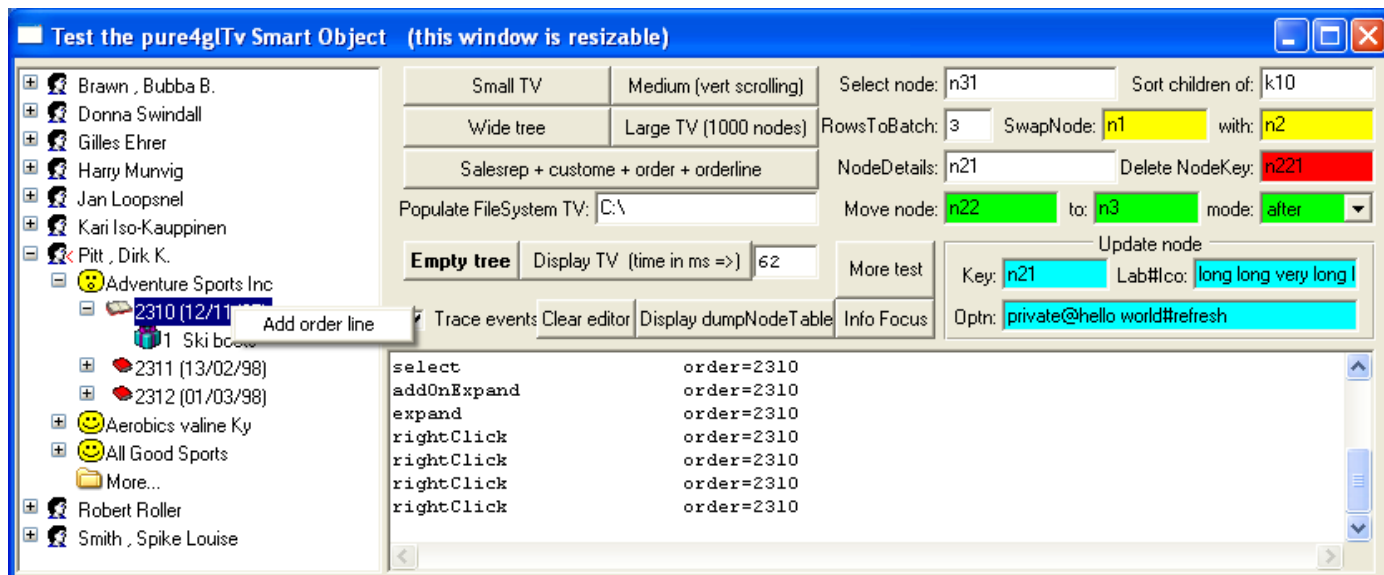


Illustration 13: Pure4glTv can build custom popup menu on the right-click event

How is this done? See the code in tvnodeEvent, it calls tvNodeCreatePopup that will just return a list of pairs of menu items and events to fire when a menu-item is chosen. Note the returned list will be passed back to the pure4glTv by tvNodeEvent with a RETURN RETURN-VALUE. Note that pure4glTv will take care of this RETURN-VALUE only if the event was rightClick, otherwise it will just ignore it.

The pure4glTv will then build the popup menu dynamically with the RETURN-VALUE. It will fire the corresponding event with the parent node key of the popup menu when the menu item is chosen.

Choose the Item. The editor shows that tvNodeEvent is fired with the expected event name of MenuAddOrderLine that was supplied in the list mentioned above.

The above example has only one menu-item, and one may wonder how to display a rule item in the popup menu. Then clear the treeview, choose the Small TV button and right click on a node. You will then get the popup menu seen in Illustration 14 with two items and a rule in between, thanks to returned string of "Add a child node,MenuAddChildNode,RULE,,Hello World,MenuHelloWorld".

Notice in the list that the item after RULE is blank. Actually, one might well put whatever here, since a rule item is just a decoration item. Note also that pure4glTv does not support sub-menus in popup menu. For more information, see the buildAndOpenPopupMenu API.

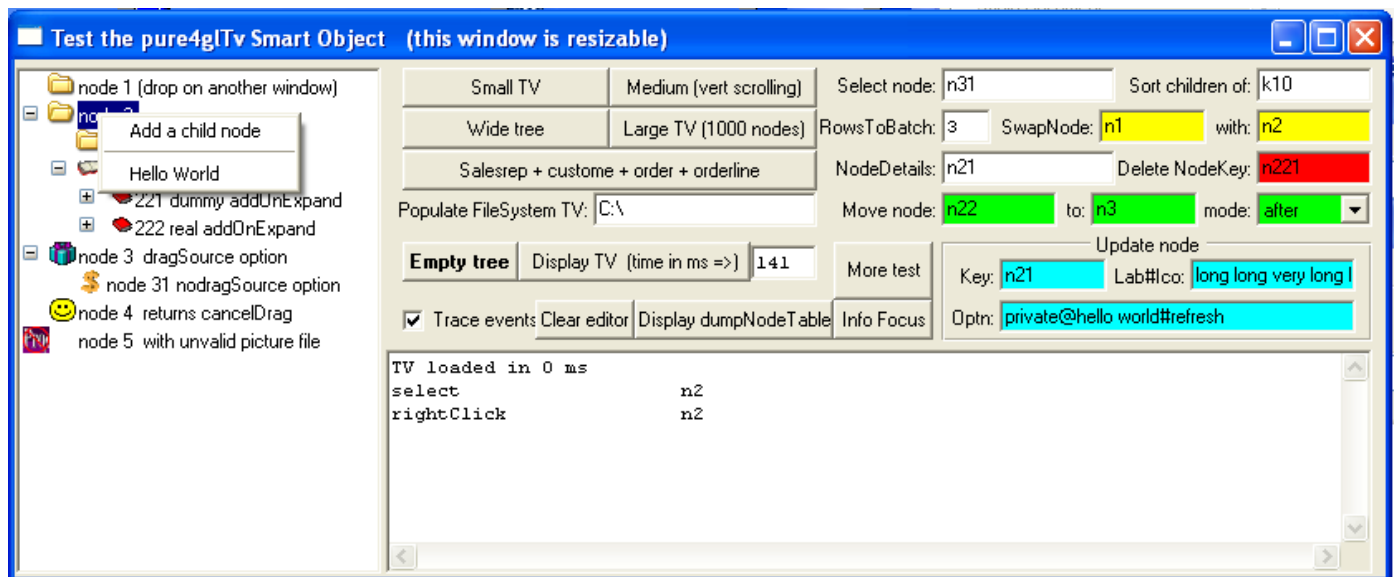


Illustration 14: A Popup menu with multiple items and a decoration rule

4.4 Testing few APIs (with the colored widgets on the right)

For more informations about the following tests, just see the code of the corresponding widget triggers. Note that the Enter key is the most used trigger in the colored widgets in order to run few common APIs, as a few tooltips say.

Load the *Small TV*, then press ENTER in a yellow *SwapNode* fill-in. This will swap *node n1* with *node n2*.

Press Enter in the *delete NodeKey* red fill-in, this will delete *node n221*.

With the green *move node* fill-in, it is possible to move any node almost anywhere in the tree. As mentioned earlier in this document, it is not possible to move a parent so it would become a child of itself. See how such an error is handled in the code.

The blue widgets give the ability to update the label or the picture or the options of a node. Read the tooltip of these widgets to understand how to use them and see the *updateNode* API for more information.

The *NodeDetails* fill-in has an interesting piece of code to get all possible information about a given node, and even obtain a handy buffer handle that will allow someone to walk through the tree with great performance. See the *getNodeDetails* API for more information.

5 A few Performance tests

The following results often show that pure 4GL things run with a very similar speed on both Linux/Wine and MS Windows as long as it does not involve the UI. Pure4gltv caches the key attributes of key widgets in a few temp-tables to avoid costly queries of attributes at the UI level. Note that the size of cached data is rather small, so the involved resources are rather small.

5.0 Results on Linux / Wine (with a 2GHz Processor)

Illustration 15 and Illustration 16 show that it takes the same time to load 1000 nodes in the tree. However the first rendering of 49 iterations in the view port costs 1.3 seconds whereas it costs only 406 milliseconds to render 17 iterations.

Note that expanding or collapsing nodes or even scrolling will cost less because the rendering widgets will be reused at best by the optimization procedure. Note it is especially interesting to set the

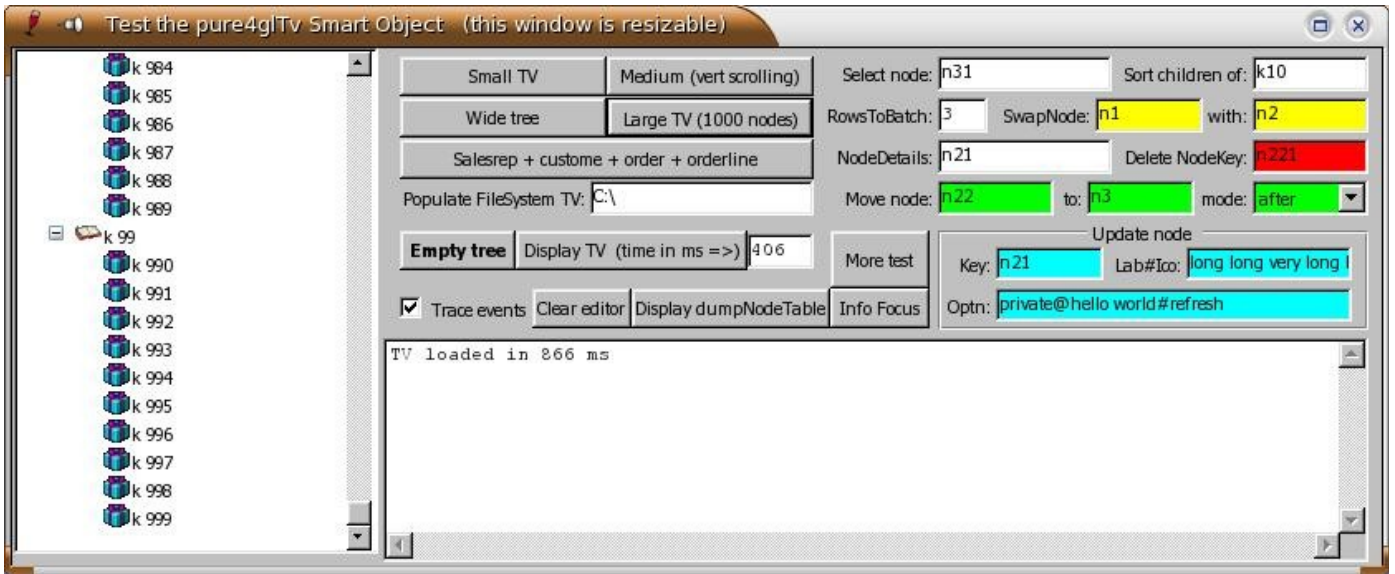


Illustration 15: On Wine 1000 nodes are added in 866 milliseconds. The rendering of 17 iterations takes 406 milliseconds.

picCacheCoef and labCacheCoef properties to a value greater than 1 (like 2 or 3) for such a large treewiew with Wine.

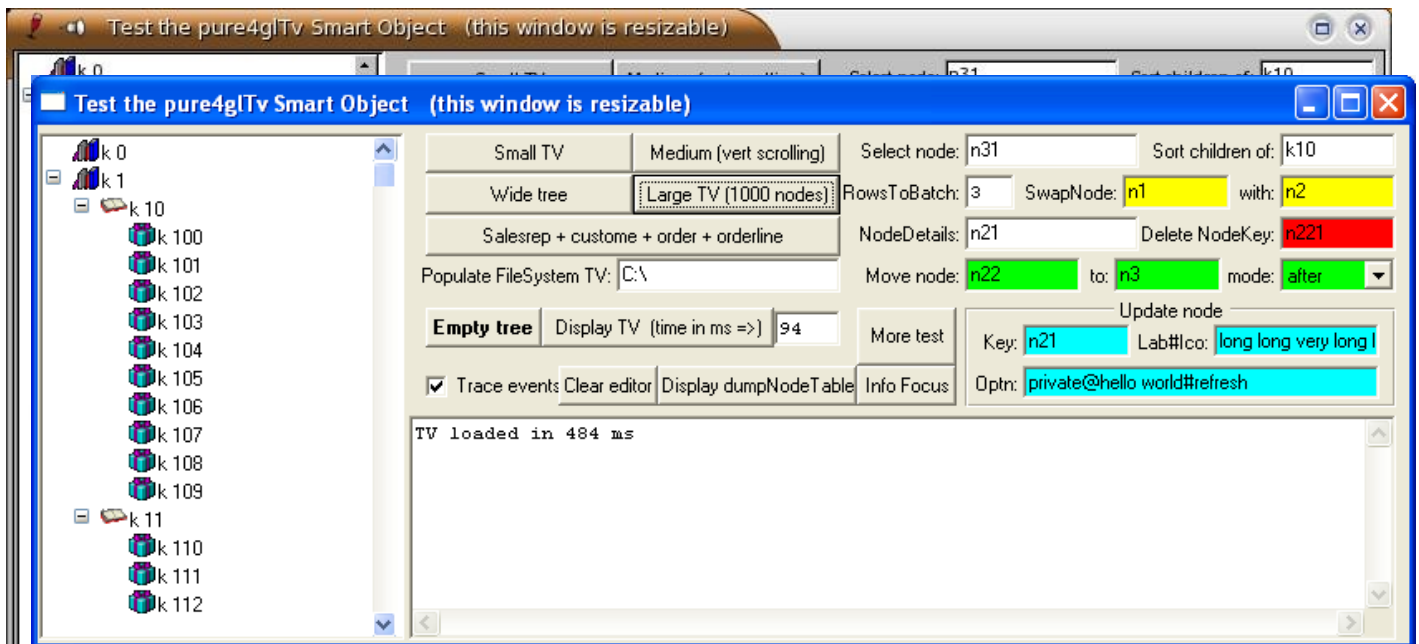


Illustration 17: On native MS Windows, 1000 nodes are added in 453 milliseconds. The rendering of 17 iterations takes 94 ms.

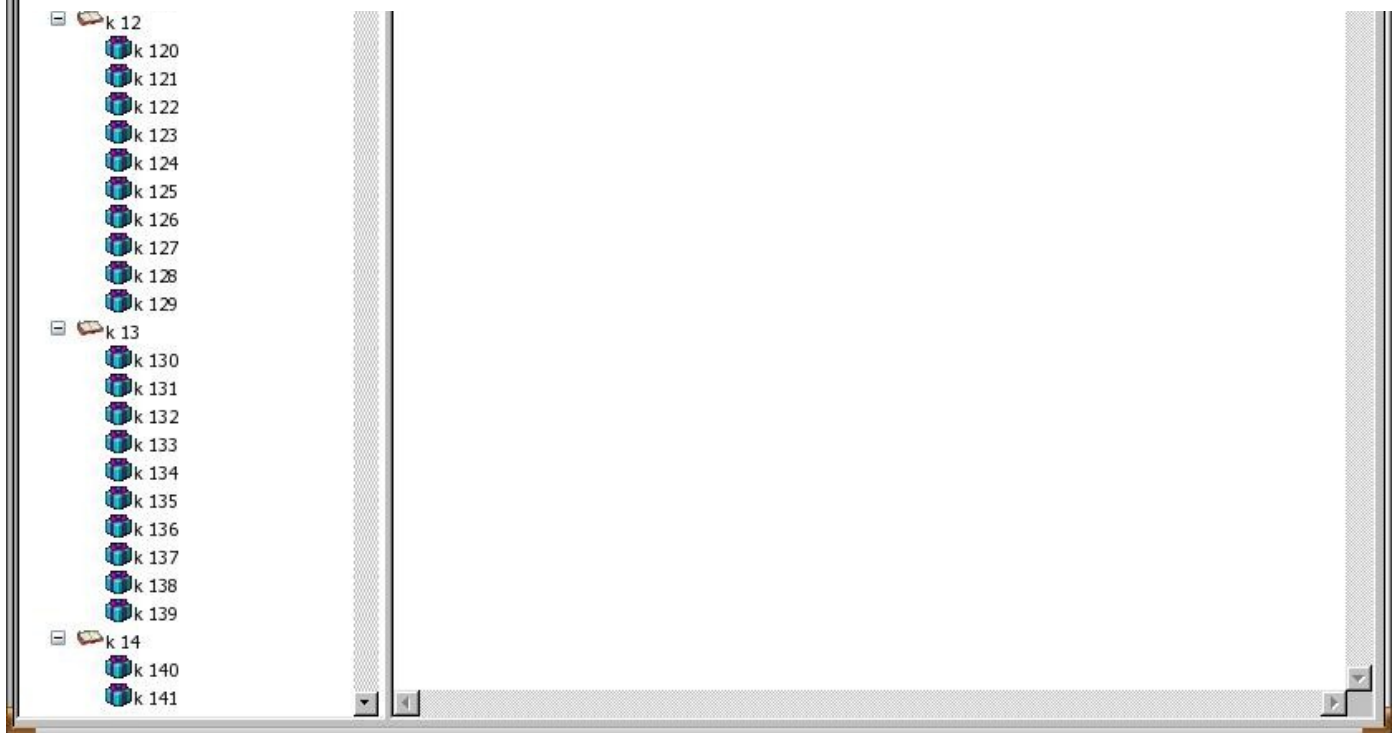


Illustration 16: On Wine. 1000 nodes still loaded in 866 milliseconds. But the rendering of 49 iterations now takes 1.3 seconds.

5.1 Results on Native MS Windows (with a 3.2GHz Processor)

As expected, performances are better on native MS Windows, not because of the addition of 1000 nodes in 453 milliseconds but because the of rendering of 17 iterations that takes 94 milliseconds.

Details to compare the results of the two configurations although the CPU used on the MS machine is much faster:

- $866\text{ms} / 3.2\text{GHz} * 2\text{GHz} = 541$ milliseconds that is rather close to 453ms. This shows that the performances on Wine are very close to the one of a native Windows machine for pure 4GL things, i.e. as long as the UI is not involved.
- On the other hand, regarding the rendering of 17 iterations: $406\text{ms} / 3.2\text{GHz} * 2\text{GHz} = 253\text{ms}$ that is quite far from the 94ms obtained on Native Windows. So in short, the display is about 2.5 slower on Wine than on native Windows.

This snapshot shows again that 1000 nodes still take the same time to be loaded. Note that the 49 iterations in the view port costs only 156 ms on MS Windows, which is really fast for a normal production.

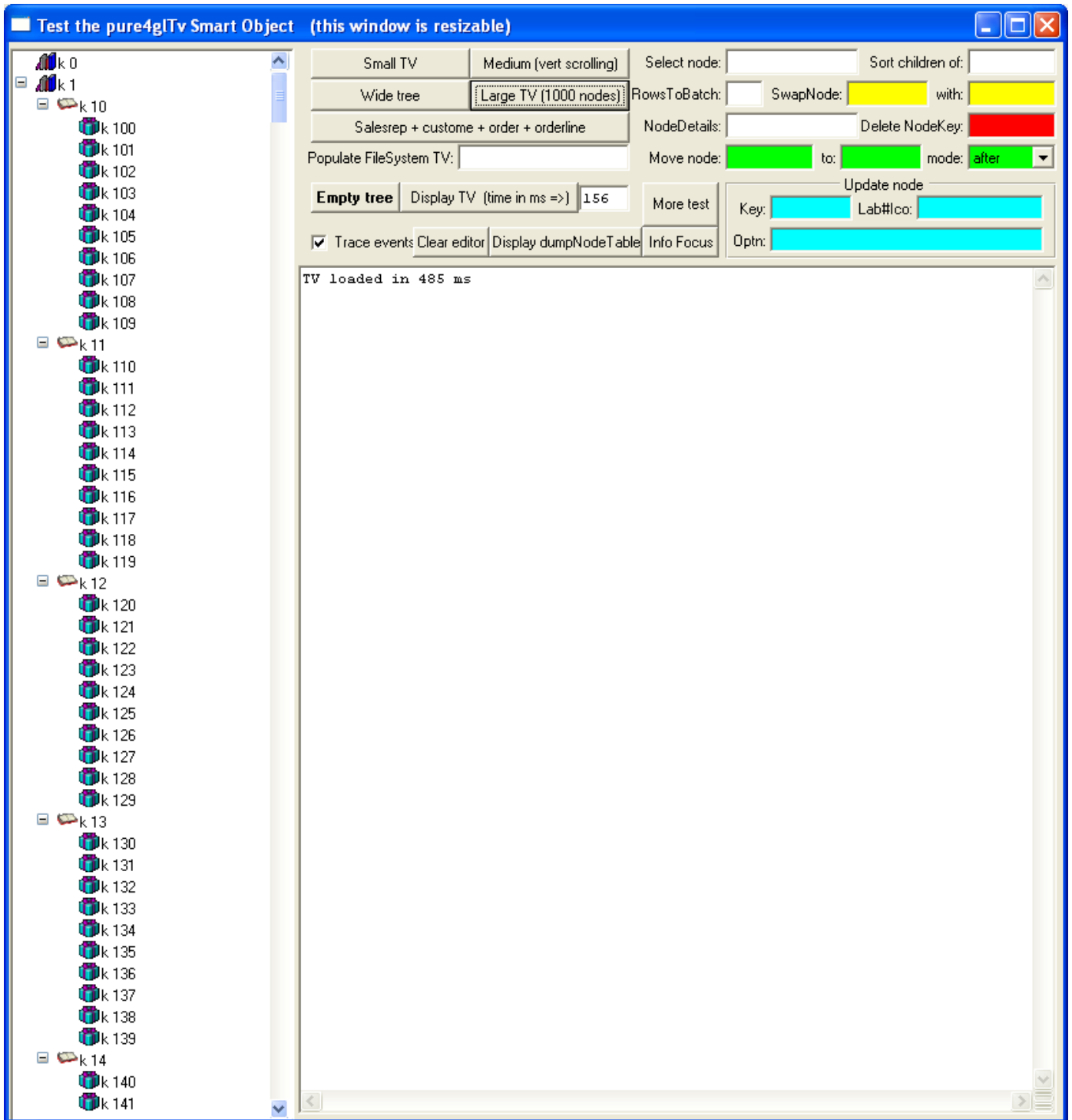


Illustration 18: Pure4glTv is very fast to refresh 49 rendering iterations on Native MS Windows

6 API documentation

The following documentation actually comes from the source code of pure4glTv. It is just a copy of the header comment for each API.

This list of APIs does not include the `get<Property>` and `set<Property>` functions. Refer to the property list and their entry in the source code for more information.

Finally, these APIs are procedures unless the name ends with parentheses `()` meaning it is a function.

6.0 addNode

```

/*-----
Purpose:      Add a node to the treeview.  A node is added to a parent node as
              a last child.  We cannot insert t between two children for now
              (easy to implement an option later...)

Parameters:

pcKe:         User key used for the node to add
              if blank, then we generate a key with "generatedKey-<bideId>>"

pcKePar:      user key of parent, or "#par=<id of parent node>"

pcLab:        label of the node

pcIco:        icon or set of image.  If blank then classic folder is used.
              => see picFileName() function to see how implement a new set of images

pcOptn:       An extendable CHR(1) separated list of options, for the node to add,
              or for the addNode API itself (such as "refresh"):
              Actually, options that are not valid for the addNode API itself
              are considered as options for the node record to be added
              Unknown options are taken as custom options and are stored in node.optn

--- node options considered when adding the node, but not kept in node.optn -----
expanded      node initially expanded

selected      Go to that node and select it at the next tvRefresh()

--- node options stored in node.optn for later use -----
addOnExpand   Handle it as a collapsed node even when it has no child
              expanding it will result in publish "tvnodeEvent" with
              event "addOnExpand" then it will expand the node

InViewPortIfPossible  In next tvRefresh() will try to make the node
              visible in the view port by scrolling the treeview
              if necessary

dragSource     When the property DragSource is set to "some", then a drag
              operation is allowed for nodes that have the option
              "dragSource"

noDragSource   Disable the drag event even when the property DragSource is set
              to "all"
              => Note also that another way to disable a drag operation
              is to make tvNodeEvent "dragBegin" RETURN "cancelDrag"

```

```

--- option for the addNode API itself (not for the node to add) -----
AddMode=after  Instead of adding the node as a child of pcKePar, the
                node is added as a next sibling of node pcKePar
                => to do that, we find the parent of pcKePar, add the node
                to it then run MoveNode to move after the wanted node
** if autoSort is set, then this option is used only to retrieve the parent node
   but the new node is added at a place that is compliant with the sort

AddMode=before  Instead of adding the node as a child of pcKePar, the
                node is added as a prev sibling of node pcKePar
                => to do that, we find the parent of pcKePar, add the node
                to it then run MoveNode to move before the wanted node
** if autoSort is set, then this option is used only to retrieve the parent node
   but the new node is added at a place that is compliant with the sort

refresh        Calls tvrefresh() at the end of the addNode process

AutoSort=yes/no 19-JUN-2007 sla new feature.  New option to superceed the autoSort TV
                option for one single node

font=n         17-OCT-2007 SLP New option to set the node text font, where n is the font
                number in the font table.

fgcolor=n      17-OCT-2007 SLP New option to set the node text color, where n is the
                colour number in the colour table.

bgcolor=n      17-OCT-2007 SLP New option to set the background colour behind the
                node text, where n is the colour number in the colour table.

tooltip=xxx    30-OCT-2007 SLP New option to set the text of the node tooltip.
-----*/

```

6.1 buildAndOpenPopupMenu

```

/*-----*/
Purpose:      build and Open Popup Menu parented by pcKey
              This API is called automatically from labLeftMouseEvent when
              the publish of tvNodeEvent "rightclick" gets a RETURN-VALUE back
              from the linked object

Parameters:
pcKey        the key of the node that is going to parent the popup menu

pcLabelsEvents  Comma separated list of item label and item event to fire
                when the menu is chosen.

Example:
"Hello world,MenuHelloWorld,RULE,,Add Customer,MenuAddCust"
=> Choosing the Item with label "Add Customer" will fire:
PUBLISH "tvNodeEvent" ("MenuAddCust" , pcMenuParentNodeKey).

Note that if an item label is set to RULE (in *capital* case),
then a the item is taken as a RULE subtype menu item

Notes:       It is a good practice to prefix the events with something like
              "Menu" or "PopupMenu" so it is easier to handle it in the
              tvNodeEvent target procedure

              I do not handle the case when a pair is missing.  In this
              situation one will hit:

```

```
ERROR Entry <entry#> is outside the range of list <list-string>. (560)
```

```
If a blank or invalid key is passed, then the menu is simply not
parented to any node (We actually parent it to the frame).
```

```
This API should normally be called from labLeftMouseEvent on
right mouse click with key of the current selected node (bellow
the mouse pointer) but nothing prevents you from calling directly
One little drawback is that it will open the menu at the current
location of the mouse pointer...
```

```
-----*/
```

6.2 collapseNode()

```
RETURNS LOGICAL
```

```
(INPUT pcNodeKe AS CHAR
,INPUT pcOptn AS CHAR) :
```

```
/*-----
```

```
    Purpose: Collapse a given node
```

```
Parameters: pcNodeKe: key of the node to collapse
```

```
    pcOptn: extendable comma separated list of options:
            refresh      Calls tvrefresh() at the end of the collapse Process
```

```
Notes: This API returns No in the two following cases:
```

- there is no node for the passed pcNodeKe
- => in this case, it also returns an ERROR
- the node to collapse is already collapsed

```
When it is a success, then it does the following:
```

```
PUBLISH "tvNodeEvent" ("collapse", bnode.ke).
RETURN YES.
```

```
-----*/
```

6.3 deleteNode

```
/*-----
```

```
    Purpose: Delete a given node, and the branch it parents if any
```

```
Parameters:
```

```
    pcKe: User key used for the node to delete
```

```
    pcOptn: extendable comma separated list of options:
            refresh      Calls tvrefresh() at the end of the deletion process
```

```
Notes:
```

```
-----*/
```

6.4 deselectCurrentNode()

```
RETURNS LOG:
```

```
/*-----
```

```
    Purpose: Deselect the node that is currently selected
```

```
Notes: This API always returns TRUE and sets the global variable gCurNode to 0
```

```
When there is really a node to deselect, then it does:
```

```
PUBLISH "tvNodeEvent" ("deselect", bnode.ke).
```

```
-----*/
```

6.5 disableObject (new on July 2006)

```

/*-----
Purpose:      Provide a standard ADM2 API to disable pure4gltv
Parameters:   <none>
Notes:       Implemented in March 2006
-----*/

```

6.6 enableObject (new on July 2006)

```

/*-----
Purpose:      Provide a standard ADM2 API to enable pure4gltv
Parameters:   <none>
Notes:       Implemented in March 2006
-----*/

```

6.7 expandBranch (new on July 2006)

```

/*-----
Purpose:      Expand all the nodes of a branch starting from a given node
Parameters:   pcKe : node key the branch to expand starts from
Notes:       Typically fired when the '*' key is pressed, but can be called
              directly
-----*/

```

6.8 expandNode()

```

RETURNS LOGICAL
(INPUT pcNodeKe AS CHAR
,INPUT pcOptn AS CHAR /*list of option like "refresh" */) :
/*-----
Purpose:      Expand a given node

Parameters:   pcNodeKe: key of the node to expand

              pcOptn: extendable comma separated list of options:
                    refresh      Calls tvrefresh() at the end of the expand Process

Notes:       This API returns No in the two following cases:
            -there is no node for the passed pcNodeKe
              => in this case, it also returns an ERROR
            -the node to expand is already expanded

            When it is a success, then it does the following:
            PUBLISH "tvNodeEvent" ("expand", bnode.ke).
            RETURN YES.
-----*/

```

6.9 emptyTree

```

/*-----
Purpose:      delete all node and display an empty tree
Parameters:   <none>
Notes:       will call tvRefresh() at the end of the process
-----*/

```

6.10 findNextNodeToShow

```

/*-----
Purpose:      Find the Next available node of a given node in the treeview
              => can be first child or next sibling, or an uncle ;) (next sibling
              of a parent or grant parent)
-----*/

```

```

Parameters: INPUT  Node ID we want to find the Next node of

            INPUT  plIgnoreChild  mode
                   YES => Ignore expanded children nodes bellow the given node
                   NO  => This mode is like finding the node in the next visible
                           iteration

            OUTPUT Node id of the wanted Next node.
                   Equals to 0 when no available next node
-----*/

```

6.11 findPrevtNodeToShow

```

/*-----
Purpose: Find the Previous available node of a given node
Similar to the findNextNodeToShow API, but to get a previous node
(in the iteration above a given node)

=> see comments in findNextNodeToShow
-----*/

```

6.12 fMainKeyEvent

```

/*-----
Purpose:
Parameters: Key function in the frame.
            Can be a navigation key or "+" or "-"
Notes:     Was designed to be called internally, but it should work fine to
            be called from anywhere, so I unchecked the PRIVATE option
-----*/

```

6.13 getNodeDetails

```

/*-----
Purpose: To obtain the details of a given node

Parameters: pcKe, node key of the node we want the details
            One can also pass "nodeId=<nodeId>" instead of a node key

Notes:     This API is rather powerful because it returns the HANDLE of a
            DYNAMIC buffer that we create for the occasion

Advantages:
            Any field in the node table is available (with dynamic programming)

            The buffer can be reused to walk though the tree with a dynamic
            FIND-FIRST() method or a Dynamic Query, with very high performance

One disadvantage however:
IT IS UP TO THE DEVELOPPER TO DELETE the dynamic buffer object
when it is not needed anymore otherwise, one may get a memory
leak.
Note that the dynamic buffer is created in the unnamed widget pool
so it will be deleted with the pure4glTreeview Object

```

Example:

```

RUN getNodeDetails IN h_pure4gltv
( INPUT mynodeKey /* CHARACTER */,
  OUTPUT hNodeBuffer /* HANDLE */).

```

MESSAGE

```

"id:"          hNodeBuffer:BUFFER-FIELD("id"):BUFFER-VALUE      SKIP
"lab:"        hNodeBuffer:BUFFER-FIELD("lab"):BUFFER-VALUE     SKIP
"ico:"        hNodeBuffer:BUFFER-FIELD("ico"):BUFFER-VALUE     SKIP
"level:"      hNodeBuffer:BUFFER-FIELD("level"):BUFFER-VALUE   SKIP
"par:"        hNodeBuffer:BUFFER-FIELD("par"):BUFFER-VALUE     SKIP
"prev-sibling:" hNodeBuffer:BUFFER-FIELD("pre"):BUFFER-VALUE   SKIP
"next-sibling:" hNodeBuffer:BUFFER-FIELD("nex"):BUFFER-VALUE   SKIP
"expanded:"   hNodeBuffer:BUFFER-FIELD("expanded"):BUFFER-VALUE SKIP
"optn:"       hNodeBuffer:BUFFER-FIELD("optn"):BUFFER-VALUE
              VIEW-AS ALERT-BOX INFO BUTTONS OK.

```

```
DELETE OBJECT hNodeBuffer.
```

```
-----*/
```

6.14 getNodeId()

RETURNS INTEGER

(pcKe AS CHAR) :

```
/*-----*/
```

Purpose: Returns node.id of node record that has node.ke = pcKe

Notes: Returns 0 if no such node

```
-----*/
```

6.15 getNodeKey()

RETURNS CHAR

(piId AS INT) :

```
/*-----*/
```

Purpose: Returns the node key node.ke of node record that has node.id = piId

Notes: Returns "" if no such node.

Note that a node cannot have a blank key, even when a blank node key was passed to addNode() to create a node, because addNode then generates a key based on the unique id of the node (see code of addNode)

```
-----*/
```

6.16 getNodeLocatedAtXY()

RETURNS CHARACTER

(ipX AS INT

, ipY AS INT) :

```
/*-----*/
```

Purpose: This API returns the node key of the node that is located on a given (x,y) in the viewport (FRAME ftv)

This API is useful to achieve a move node by drag and drop (node dropped in the treeview)

Notes: If there is no node located at the given (x,y) then an empty string is returned

```
-----*/
```

6.17 getNodeParentKey()

RETURNS CHAR

(pcKe AS CHAR) :

```
/*-----*/
```

Purpose: Returns key of parent node

Notes: Returns blank if the node has no parent (first level in the tree)

Returns ERROR if the node does not exist

```
-----*/
```

6.18 *getSelectedNodeKey()* (new on July 2006)

```

/*-----
Purpose:  returns the node key of the current selected node (or last
          selected node)
Notes:   If there is no selected node, then we return unknoww value.
-----*/

```

6.19 *goToNode()*

```

RETURNS LOGICAL
( ipGoToNode AS INT,
  pcMode AS CHAR ) :
/*-----
Purpose:  Go to a given node.

Parameters:

          ipGoToNode  node ID we want to go to.

          pcMode:
          - blank (default), will do nothing if the node is already in the viewport.
          - "top" then will force a refresh to put the node a the top of the viewport

Notes:   We do not test if the node is valid for performance reasons. Up to
          you to call this API for a node that exists
-----*/

```

6.20 *lastVisibleNode()*

```

RETURNS INTEGER
( /* parameter-definitions */ ) :
/*-----
Purpose:  Return node.id of last 'visible' node (available in last non collapsed segment)

Notes:   Returns 0 if tree is empty

          We start from top level and seek last child of last child of last child...
          and stop if we find a non expanded guy on the way
-----*/

```

6.21 *loadDemoTv*

```

/*-----
Purpose:   Guess what...
Parameters: <none>
Notes:    Called internally at design time in order to show a cute little
          treeview
-----*/

```

6.22 *moveNode*

```

/*-----
Purpose:   Move a given node to wherever

Parameters: pcNodeKey  Key of node to move

          pcToKe       Key of relative node to move to (after it, before it, last child of
it)

          pcMode       One of the 3 following values
                       "after"
                       "before"
-----*/

```



```
"parent"
```

```
pcOtpn    extendable comma separated list of options
          "refresh" => calls tvRefresh() at the end of the process
```

```
Notes:    IF pcMode = "parent" and toNode.Ke is blank THEN it means
          'move it so it becomes the last guy at level 0'
          => In this case, the buffer toNode will hold no record
```

```
-----*/
```

6.23 nodeAtVirtualIter()

```
RETURNS INTEGER
  (VIterToGo AS INT) :
```

```
/*-----*/
```

```
Purpose:    Returns the node ID of the node located at a given virtual iteration.
          This is quite important to handle scrolling.
```

```
This API returns a node ID and not a node key because it was designed
to be mainly used internally. Anyway, it can be used from the outside
without any problem
```

```
What I call a virtual iteration is the 'line' number of a node if the
Treeview was big enough to show all the nodes in a very tall viewport
with first top root node at line number 1.
```

```
Notes: Exceptions:
```

```
Returns 0 when no node in the tree
```

```
Returns -1 when VIterToGo > gExpChildren
```

```
Returns bellow -1 when unexpected error that should never happen since it
would mean the tree is corrupted (see in the code)
```

```
-----*/
```

6.24 nodeInCollapsedBranch()

```
RETURNS LOGICAL
  (ipNode AS INT) :
```

```
/*-----*/
```

```
Purpose:    Returns yes if a given node (based on node ID) is in a collapsed
          branch, else no
```

```
Notes:    In the case of an invalid node, it RETURNS ERROR NO
```

```
-----*/
```

6.25 picFileName()

```
RETURNS CHARACTER
  (cCode AS CHAR,
   ico AS CHAR) :
```

```
/*-----*/
```

```
Purpose:    Returns a picture file name for a given code and icon/picture file.
```

```
Notes:    Note that ico contains the file name path prefixed by a path, which
          might be relative
```

```
Convention to implement your own set of pictures, called <icoFile> here
```

```
With a Plus/Minus on the left
```

```
noChild   <icoFile>NoSign
```

```
expanded  <icoFile>Minus
```

```
collapsed <icoFile>Plus
```

```
Without Plus/Minus on the left
```

```
noChild   <icoFile>
```

```
expanded  <icoFile>Open
```

```
collapsed <icoFile>
```

```
-----*/
```

6.26 *resizeObject*

```

/*-----
Purpose:      standard ADM2 API
Parameters:   height and width in char units

Notes:       It is up to the developer to
              1) call this API
              2) query the resizeVertical and resizeHorizontal instance
                 properties to supply the expected values

              If a parameter is ? or 0, then it means that the width or
              height has to remain unchanged

Example in window-resize of a container window:

{get resizeVertical lresizeVertical h_pure4glTv}.
IF lresizeVertical = ? THEN lresizeVertical = YES.
{get resizeHorizontal lresizeHorizontal h_pure4glTv}.
IF lresizeHorizontal = ? THEN lresizeHorizontal = YES.

iTvWidth = FRAME fMain:COL - 1.7. /* the treeview is on the left border */

iVerticalGap = SELF:HEIGHT-CHAR - FRAME fContainer:HEIGHT-CHARS.
iHorizontalGap = SELF:WIDTH-CHAR - FRAME fContainer:WIDTH-CHARS.

[...] /* resizing of frame in the smart window */

RUN resizeObject IN h_pure4glTv
  (IF lresizeVertical THEN SELF:HEIGHT-CHARS - 0.2 ELSE ?
  ,IF lresizeHorizontal THEN iTvWidth + iHorizontalGap ELSE ?).

[...]
-----*/

```

6.27 *selectNode()*

```

RETURNS LOGICAL
(pcNodeKe AS CHAR) :
/*-----
Purpose:      Select a the node of a given key

Notes:       For now, if we select a node that is outside of the view port, then
              we try to scroll to it
-----*/

```

6.28 *sortChildren*

```

/*-----
Purpose:      Sorts (or resorts) all the children of a parent by their label

Parameters:   pcKe  key of the parent node

              pcOptn is an extendable comma separated list of option:
                  refresh => call tvRefresh() at the end of the sorting process
-----*/

```

6.29 *swapNode*

```

/*-----
Purpose:      Swap a node with another.

              We do not touch the children nodes, which remain attached to their
              parent, so this API could also have been called swapBranches

```

Parameters: See definition block

Notes: It is more efficient to call this API than calling moveNode multiple times.

6.30 topNode()

RETURNS INTEGER

(/* parameter-definitions */) :

```
/*-----*/
Purpose: returns node.id of node at the top of the tree
Notes: Note this node cannot be in a collapsed branch
-----*/
```

6.31 updateNode

```
/*-----*/
Purpose: Update the label, or the icon, or a list of option(s) of a given node.
Parameters:          pcKe: key of the node to update

pcfieldName: Comma separated list of fields
              => for now the complete list is limited to "lab,ico"
              the idea is to keep it flexible for possible future needs

pcFieldValues: CHR(1) separated list of values for the fields listed
              in pcfieldName.

pcOptn: BEWARE, this one is not only about options for the API
        itself, but especially to update the options of a node
        in the node.optn field, as explained bellow:

pcOptn contains a CHR(1) separated list of option to add, update
or remove from the node.optn record to update.
=> If the option is not in node.optn yet, then add it

=> If the option is of the form 'name' + CHR(2) + 'value' and there
is already an option with same 'name' + CHR(2) in node.optn,
then update the value

=> If the option is in node.optn, but pcOptn has it prefixed with
"!" (exclamation mark) then it is *removed* from node.optn
example "!private" will remove "private<CHR(2)>hello world"

See the complete list of standard node options in the header
comments of the addNode API

If "refresh" is passed in the the option list, then a tvRefresh() is
called at the end of the update process (I did not want to create yet
another parameter just for this need...)
=> SO "refresh" IS AN EXCEPTION (don't use it as a custom node option)

Notes: Note that this API does not only modify node.lab and node.ico, but
also takes care of the virtual width of the node to notify the
treeview if it has to enlarge or shrink its virtual width
-----*/
```

6.32 updateNodeWidth

```
/*-----*/
Purpose: Update the width of a node, by taking care of the virtual width of
```

the viewport

Parameters: see definition block

Notes: This procedure was designed to be called internally, but it should work OK when called from the outside, so I unchecked the PRIVATE option

6.33 tvRefresh()

```
RETURNS LOGICAL
  (/* char def */) :
```

```
/*-----*/
Purpose: Refreshes the view port of the treeview. Used to display the current
         view port
```

Notes: Big change on 01-Nov-2004: we always refresh from iteration 1
Before, we could refresh starting from the middle of the viewport
(think of collapsing a node in the middle) for performance sake,
However it is far simpler and easier to maintain by always refreshing
the all view port and the impact on performance is ridiculous thanks
to new optimizeWine/MS procedures that naturally reuse the rendering
widgets at best.

6.34 tvScroll()

```
RETURNS LOGICAL
  ( ipScrollBy AS INT,
    scrollAsMuchAsPossible AS LOG) :
```

```
/*-----*/
Purpose: Scroll view port by ipScrollBy
         positive means scroll up    (going down!!!)
         negative means scroll down  (going up !!!)
```

Notes: Description of the scrollAsMuchAsPossible parameter:
Scenario: We ask to scroll 6 up but there are only 5 iterations
above the top so the maximum we can scroll up is actually 5:
-if scrollAsMuchAsPossible IS set, then we scroll up by 5
-if scrollAsMuchAsPossible is NOT set, then we do NOT scroll
at all and we RETURN NO

Two little exceptions when we return NO for any value of scrollAsMuchAsPossible:
1) If we are already at the very bottom and try to scroll up
2) We are already at the very top and we try to scroll down

6.35 widgetsAt

```
/*-----*/
Purpose: Return the list of widget handles of the widget(s) that are located
         on a given X and Y
```

Parameters: see definition block

Notes: This procedure was designed to be called internally, but it should work OK when called from the outside, so I unchecked the PRIVATE option

7 Many Thanks to

- Jurjen Dijkstra who provided the window.i windows.p winfunc.i and winfunc.p on <http://www.global-shared.com>

- My wife Amandine who has patiently put up with the time I spent on this project at home.
- Richard Tardivon who gently offered to do some beta testing
- Jean Richert because it cannot harm to give a few thanks to ones manager ;)
- Vladimir Zalda for giving me motivation – far beyond what he may expect..
- Sasha Kraljevic for suggesting to support of popup menus and the drag and drop. These were great stuff
- Nicolas Andricq for his involvement in pure4glTv to restructure the support of multiple skins, such as the Royal XP theme, and for his special care about the vertical scrollbar 1 pixel issue.
- Simon L Prinsloo and Dries Feys for the support of fonts and colors at the node level