rTree v.2.0
Quick Reference.

Commands.

To create a new rTree type "**new_rTree**" into the message box and hit return.

You **dispatch commands** to the rTree control such as:

new_node - Creates a new node in the rTree control that the new_node command is dispatched to.

move_node_ID - Moves a node in the rTree control that the new_node command is dispatched to.

move_node - Moves a node in the rTree control that the new_node command is dispatched to.

delete_node - Deletes a node or nodes from the rTree control that the delete_node command is dispatched to.

delete_node_ID - Deletes a node or nodes from the rTree control that the delete_node_ID command is dispatched to.

delete_nodeContainers - Deletes the groups that holds the controls of a node.

delete_nodeData - Deletes the node data from the rTree control that the delete_nodeData command is dispatched to.

scan_node_ID – Scans a node to get its chidlren from a filesystem using the node *sourcePath* property.

delete_treeData - Deletes all data from the rTree control that the delete_treeData command is dispatched to.

render_tree - Processes all tree and node data and renders the rTree control that the render_tree command is dispatched to.

updateTree – Renders the rTree control without processing the array data.

updateGeometry – Up dates the rTree controls geometry.

Example:

```
dispatch "new_node" to control "Tree" with "as root 1"
dispatch "new_node" to control "Tree" with "as root" && theNumber
dispatch "new_node" to control "Tree" with "as child – 1 of node id 1"
dispatch "move_node_ID" to control "Tree" with tID && "as child 2 of node id" && tParentID
dispatch "delete_node_ID" to control "Tree" with 1,3,5,7
dispatch "delete_node" to control "Tree" with "myNodeName"
dispatch "delete_nodeData" to control "Tree"
dispatch "scan_node_ID" to control "Tree"
dispatch "updateGeometry" to control "Tree"
```

Tree Properties.

Examples:

```
set the autoScrollbar of control "Tree" to true
```

autoScrollbar
Specifies whether the scrollbars of an rTree control is shown automatically when needed or not.

autoStartUsing
Specifies whether the rTreeEngine will put itself into the message path as a backscript or not.

betweenNodesDelta
Specifies the distance in pixels the pointer must be within from a node line edge during a drag & drop for the drop to be between two nodes. If set to 0 drop will not be between nodes at all.

clickIcon (read only)
Reports the ID of the node icon the user last clicked in an rTree control.

clickNode (read only)
Reports the ID of the node the user last clicked in an rTree control.

copyBetweenTrees
Specifies if dragged nodes are copied or moved from the source rTree control. Default setting is false.

depthDelta
Sets the number of pixels that each depth of the tree structure is indented with. Setting it to 0 makes all nodes left aligned regardless of their depth.

dontChangeNodeIDs
If set to true, the nodeIDs of dragged nodes does not change when dropped into another rTree control. Default setting is false.

editNodeOnDoubleClick
Specifies whether the Node name can be edited by doubleclicking it or not.

expandOnHilite
Specifies whether hiliting a node automatically expands and scrolls that node into view or not. Default setting is true.

fileIcon
Sets the **ID** of an image used as the second icon of a node if the node represents a file.

firstIconCollapsed
firstIconExpanded
secondIconCollapsed
secondIconExpanded

thirdIconCollapsed
thirdIconExpanded
Specifies an image that new nodes inherit if no icon has been set for the node itself.

hilitedNodeIDs
Specifies the IDs of the selected nodes in an rTree control.

hiliteOnMouseUp
Toggles if nodes hilites on mouseUp or mouseDown.

lastNodeID (read only)
Reports the **ID** of the last node created.

mouseNode (read only)
Reports the ID of the node that the mouse pointer is over in an rTree control.

newNodeOnReturn
Returns true or false. If true, hitting the return key in an rTree control creates a new node.

nodeData
The array that holds all node data.

nodeIDs (read only)
The nodeIDs property is a list of all node IDs in the rTree control, one ID per line and in the order the nodes would be rendered in the control if all nodes were expanded and visible.

rootNodeIDs
Sets the root node IDs of an rTree control.

showInvisibleNodes
Shows nodes even if their visible property has been set to false.

shownNodeIDs
Reports the nodeIDs actually shown to the user.

treeType
The treeType of an rTree control can be set to **empty** or "**filesystem**". If set to "filesystem" the properties, behavior and apperance of the rTree control is adopted to suite having a filesystem as the source of data for the rTree control. A filesystem is for example the file structure on a hard disk.

useContextMenu
Toggles if the built in right click context menu is used or not.

useConnectionLines
Toggles if connection lines between the nodes is shown or not.

useFirstBadge
Toggles if the firstBadge is used for the nodes or not.

useMomentumScroll
Toggles if momentum based scroll is used or not. Default is false.

useDragIndicator
Toggles if the built in drag indicator is shown during drag and drop or not.

useFirstIcon
Toggles if the firstIcon is used or not.

useSecondIcon
Toggles if the secondIcon is used or not.

useThirdIcon
Toggles if the thirdIcon is used or not.

visibleNodeIDs
The visibleNodeIDs property is a list of node IDs of visible nodes, one ID per line, and in the order the nodes is rendered in the rTree control.

Node Properties.

Example:

```
set the textStyle_of_node_ID_[1] of control "Tree" to "bold,italic,underline"
```

children
Sets the children of a node in an rTree control.

depth
Specifies the how many levels deep a node is in an rTree control.

expanded
Specifies if a node in an rTree control is expanded or collapsed.

family
Reports a list with the IDs of the family nodes including the ID of the node that owns the family an rTree control. One ID per line.

firstIconCollapsed
firstIconExpanded
secondIconCollapsed
secondIconExpanded
thirdIconCollapsed
thirdIconExpanded
Specifies an image of a node. Setting the **icon** property of a specific node overrides the **icon** property of the rTree control.

ID
Reports the unique ID number assigned to a node.

IDpath
Reports the path to the node expressed as node IDs separated by slashes.

indent
Determines the indentation of the node in an rTree control.

name
Specifies the name of an node in an rTree control.

nodeID
Reports the ID of a node of a certain line of the visible nodes of the rTree control.

nodeLine
Reports the line number of a node with a certain ID of the visible nodes in the rTree control.

number
Specifies a nodes position within a rTree control.

parent
Reports which node is next in the node hierarchy.

sourcePath
Reports the path to the data source of the node expressed as the path items of the data source path separated by slashes.

textColor
Specifies the color of the node text.

textFont
Specifies the font face of text of a node.

textShift
Specifies how far the text of a node is shifted up or down from its baseline.

textSize
Specifies the point size of text of a node.

textStyle
Specifies the style or styles applied to text of a node.

useFirstIcon
useSecondIcon
useThirdIcon
Specifies if the first icon (leftmost) of a node should be displayed in the rTree control or not.

visible
Specifies whether an node can be seen or is hidden.

Custom Properties.

Using a custom Node property is a very good way to:
• associate data with a specific Node
• save the data with the Node in the stack file
• access the data quickly

You can store the content of a field in an rTree Node like this:

```
set the myFieldContent_of_node_ID_[tNodeID] of control "Tree" to the htmlText of field "myContentField"
```

And when the user clicks the Node, the content can be displayed in a field again with this handler in the rTree controls script:

```
on nodeDown theButton theNodeID theLongTreeID
```

```
  set the htmlText of field "myDisplayField" to the myFieldContent _of_node_ID_[tNodeID] of me
  pass nodeDown
end nodeDown
```

You can use a URL to store a file's content in a custom Node property:

```
set the myPicture_of_node_ID_[tNodeID] of control "Tree" to URL "binfile:mypicture.jpg"
```

Messages.

The nodes in the rTree control can be seen as virtual LiveCode objects, contained within the rTree. As for any other object in LiveCode, the nodes can receive mouse triggered events such as:

nodeDown
nodeUp
nodeDoubleDown
nodeDoubleUp
nodeEnter
nodeLeave

Other messages received by the rTree control includes:

newNode
deleteNode
nodeCollapsed
nodeExpanded
startRenderTree
finishedRenderTree
startScanNode
finishedScanNode

Example:

```
on nodeDown
  set the expanded_of_node_ID_[7] of me to true
  dispatch "renderTree" to me
end mouseDown
```

Important! If you use the following mouse event handlers in your rTree control script you must pass the message for rTree to work correctly:

mouseMove
mouseDown
mouseUp
mouseDoubleDown
mouseDoubleUp
mouseEnter
mouseLeave

Example:

```
on mouseDown
  set the expanded_of_node_ID_[7] of me to true
```

```
  dispatch "renderTree" to me
  pass mouseDown # Pass the message for rTree to work correctly.
end mouseDown
```

Building an rTree from an array.

If you work with larger amounts of data, it is useful, instead of creating nodes by script to prepare the whole tree in one go and then only render it once.

To do so, you can use an array to represent the datastructure. This lesson you have an rTree group prepared on your card. We will assume it is called "myTree". Let's get started. The array you will be building is multi dimensional. If you refer to a node property, it will contain two keys. If you refer to a control in the group build from your node template it will contain 3 keys.

The first Key in the array is ALWAYS the node ID you are referring to, with one exception. A special case is the key with the name "0". This one refers to general tree properties, like rootNodeIDs and properties that will be inherited by all other nodes, unless they are overloaded by a key for said node.
How the other keys for your array look like depends on what you need to do. If your array entry is 2 dimensional e.G.

put true into tTreeArray["RootNode"]["expanded"]

You are working on properties that are valid for the whole node. You even can set your own custom properties there:

put the seconds into tTreeArray["RootNode"]["timeCreated"]

Setting a custom property has no visible effect on the node, however it can be read at a later point in time, so it might be quite useful to have those.

If yo want to refer to a single control inside the group that makes up your node, your array entry will have 3 keys. The first key is the ID of the node again. The second key is the name of the control in the nodeGroup you want to adress. Look at the node template group of your rTree if you want to see which ones are available by default, or add your custom controls as needed there. We will be working with a vanilla node template in this lesson.

put true into tTreeArray["Rootnode"]["firstBadge"]["visible"]

will show the right badge graphic in the node group.

The main advantage when you are working with an array for rTree, is that the order of creation of nodes is not much of importance. The order in which the nodes are rendered are governed by a list of node IDs. For root nodes it is:

tTreeArray["0"]["RootNodeIDs"]

which holds a cr delimeted list of all nodes at root level, ordered from top to bottom.
Children of a single node and their order is governed by the children property of said node:

tTreeArray["NodeID"]["children"]

again, this is a cr delimited list, ordered from top to bottom.

A script that contains all of these elements can look like this:

I am assuming you have an rTree group called "myTree" on the card. Add a button and paste the following script:

```
on mouseUp
  buildTree
end mouseUp

on buildTree
  local tTreeArray
  /*
  The following block holds general Tree properties
  Specify a node ID of 0 in the first key of the array
  and reference the property you want to set in the second key of the array
  */

  put false into tTreeArray["0"]["useContextMenu"] -- We do not want a contextmenu
  put true into tTreeArray["0"]["useConnectionLines"] -- We want connection lines between the nodes
  put false into tTreeArray["0"]["useThirdIcon"] -- we do not want to display a thirdIcon in this case

  /* Create a root node */
  put "Rootnode" into tTreeArray["0"]["rootNodeIds"]    -- this creates the first rootNode

  /*if you want to add another root node uncomment the next line*/

  -- put cr & "Rootnode 2" after tTreeArray["0"]["rootNodeIds"]

  /*
  Next we want to add 10 children as leafs to the root node.
  Leafs are nodes that itself have no further children
  */

  repeat with i=1 to 10
    put 0 into tTreeArray[("child"&&i)]["firstIconCollapsed"]
    put 0 into tTreeArray[("child"&&i)]["firstIconExpanded"]  -- these are leaves with no children
    put "2537" into tTreeArray[("child"&&i)]["secondIconCollapsed"]
    put false into tTreeArray[("child"&&i)]["expanded"]
    put ("I am child"&&i) into tTreeArray[("child"&&i)]["label"]
    if tTreeArray["Rootnode"]["children"] is empty then
      put ("child"&&i) into tTreeArray["Rootnode"]["children"]
    else
      put cr & ("child"&&i) after tTreeArray["Rootnode"]["children"]
    end if
  end repeat

  /* We want to add 10 more nodes that in return each shall have children */

  repeat with i=11 to 20
    put 2194 into tTreeArray[("child"&&i)]["firstIconCollapsed"]
    put 2195 into tTreeArray[("child"&&i)]["firstIconExpanded"]  -- these are leaves with no children
    put "2198" into tTreeArray[("child"&&i)]["secondIconCollapsed"]
    put "2197" into tTreeArray[("child"&&i)]["secondIconExpanded"]
    put false into tTreeArray[("child"&&i)]["expanded"]
    put ("I am child"&&i) into tTreeArray[("child"&&i)]["label"]
    if tTreeArray["Rootnode"]["children"] is empty then
      put ("child"&&i) into tTreeArray["Rootnode"]["children"]
    else
      put cr & ("child"&&i) after tTreeArray["Rootnode"]["children"]
    end if
```

```
    /*now append children of each child node */

    repeat with j=1 to 20
      put 0 into tTreeArray[("child"&&i&&j)]["firstIconCollapsed"]
      put 0 into tTreeArray[("child"&&i&&j)]["firstIconExpanded"]  -- these are leaves with no children
      put "2537" into tTreeArray[("child"&&i&&j)]["secondIconCollapsed"]
      put "2537" into tTreeArray[("child"&&i&&j)]["secondIconExpanded"]
      put false into tTreeArray[("child"&&i&&j)]["expanded"]
      put ("I am child"&&j&&"of child"&&i) into tTreeArray[("child"&&i&&j)]["label"]
      if tTreeArray["child"&&i]["children"] is empty then
        put ("child"&&i&&j) into tTreeArray["child"&&i]["children"]
      else
        put cr & ("child"&&i&&j) after tTreeArray["child"&&i]["children"]
      end if
    end repeat
  end repeat

  /*  the fun part is, your work does not necessariely need to be in order
  as long as you know the key for the current node
  */

  put "Did you know I am a leaf?" into tTreeArray["child 12 1"]["text of line 2"]
  put "dark green" into tTreeArray["child 12 1"]["textColor of line 2"]
  put true into tTreeArray["child 12 1"]["expanded"]
  put true into tTreeArray["child 12"]["expanded"]

  /* with a little trickery, you can do rather cool things.
  remember that you can refer to ANY control in the nodegroup ( same structure as the nodeTemplate)
  if you have 3 keys in your array */

  put true into tTreeArray["Rootnode"]["firstBadge"]["visible"]
  put the number of lines of tTreeArray["Rootnode"]["children"] into tTreeArray["Rootnode"]["firstBadge"]["label"]
  put "light blue" into tTreeArray["Rootnode"]["firstBadge"]["backColor"]
  put "blue" into tTreeArray["Rootnode"]["firstBadge"]["foreColor"]
  dispatch "nodeData" to grp "myTree" with tTreearray
  dispatch "renderTree" to grp "myTree"
end buildTree
```

If the way you aquire your data is dezentralized and you do not know the structure of the final tree in advance, the array way of setting up an rTree is the way to go. It is also advised to use it if you are handling larger amounts of data, as you do not have the overhead of calling many handlers to build the tree.