TreeView Xtra Help

For up-to-date information please visit our web site:
xtras.tabuleiro.com

TREEVIEW XTRA HELP: INSTALLATION

The installation procedure is slightly different depending on the version of Director and platform used. Make sure you have administrative rights to create files in the directory where Director is installed on your system.

WINDOWS	MACINTOSH
Director 11	Director 11
Director MX 2004	Director MX 2004
Director MX	Director MX
Director 8.5	Director 8.5

TREEVIEW XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR 11

INSTALLING THE XTRA ON WINDOWS - Director 11

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "TreeView" on your machine. To install the Xtra, just copy the file Windows\TreeView.x32 to the Director 11 XTRAS folder. If your copy of Director 11 is installed at the default location, the Windows Xtra file will be located at:

C:\Program Files\Adobe\Adobe Director 11\Configuration\Xtras\TreeView.x32

Now you need to install the files necessary for creation of cross-platform projector for Mac OSX. Go back to the "TreeView" directory where the Xtra files were unpacked. Open the **Mac Universal** directory. Now copy the file "TreeView.cpio" to the "Configuration\Cross Platform Resources\Macintosh\Xtras" directory used by Director 11. In a default installation of Director this file will end up at the following location:

C:\Program Files\Adobe\Adobe Director 11\Configuration\Cross Platform Resources\Macintosh\Xtras\TreeView.cpio

Finally, you need to edit the xtrainfo.txt file to include information about TreeView. This information is used by the Shockwave and cross-platform publishing features in Director 11, to locate the files needed when assembling the Mac OSX version of your projector. The xtrainfo.txt file is located by default at:

C:\Program Files\Adobe\Adobe Director 11\Configuration\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://download.tabuleiro.com/packages/TreeView/4/TreeView"]
```

You may want to customize this line in the future, to instruct Director to download TreeView Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.

TREEVIEW XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR MX 2004

INSTALLING THE XTRA ON WINDOWS - Director MX 2004

If you have not done so, we recommend updating to Director MX 2004 version 10.1 before installing the Xtra. This will allow creation of projectors for Mac Classic and OSX (Director MX 2004 without the update can only create cross platform projectors for OSX.)

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "TreeView" on your machine. To install the Xtra, just copy the file Windows\TreeView.x32 to the Director MX 2004 XTRAS folder. If your copy of Director MX 2004 is installed at the default location, the Windows Xtra file will be located at:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Xtras\TreeView.x32

Now you need to install the files necessary for creation of cross-platform projector for Mac OSX. Go back to the "TreeView" directory where the Xtra files were unpacked. Open the **Mac Carbon** directory. Now copy the files "TreeView.data" and "TreeView.rsrc" files to the "Configuration\Cross Platform Resources\Macintosh\Xtras" directory used by Director MX 2004. In a default installation of Director these files will end up at the following locations:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Macintosh\Xtras\TreeView.data

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Macintosh\Xtras\TreeView.rsrc

If you are running Director MX 2004 10.1, you can also install the files necessary for creation of cross-platform projector for Mac Classic. Again, go back to the "TreeView" directory where the Xtra files were unpacked. Open the **Mac Classic** directory. Now copy the files "TreeView.data" and "TreeView.rsrc" files to the "Configuration\Cross Platform Resources\Classic\Xtras" directory used by Director MX 2004. In a default installation of Director these files will end up at the following locations:

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Classic\Xtras\TreeView.data

C:\Program Files\Macromedia\Director MX 2004\Configuration\Cross Platform Resources\Classic\Xtras\TreeView.rsrc

Finally, you need to edit the xtrainfo.txt file to include information about TreeView. This information is used by the Shockwave and cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the OSX and Classic versions of your projector. The xtrainfo.txt file is located by default at:

C:\Program Files\Macromedia\Director MX 2004\Configuration\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://download.tabuleiro.com/packages/TreeView/4/TreeView"]
```

You may want to customize this line in the future, to instruct Director to download TreeView Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.

TREEVIEW XTRA HELP: INSTALLATION: WINDOWS - DIRECTOR MX AND 8.5

INSTALLING THE XTRA ON WINDOWS - Director MX and Director 8.5

Decompress the installation .zip file. This will unpack the Xtra, documentation and sample files to a folder named "TreeView" on your machine. To install the Xtra, just copy the file **Windows\TreeView.x32** to the Director 8.5 or Director MX XTRAS folder. If you have previously installed an older copy of the Xtra make sure to remove or replace it.

These are the default locations of the Xtras folder for each application:

Director 8.5- C:\Program Files\Macromedia\Director 8.5\Xtras

Director MX- C:\Program Files\Macromedia\Director MX\Xtras

Finally, you need to edit the xtrainfo.txt file to include information about TreeView. This information is used by the Shockwave publishing features in Director. The xtrainfo.txt file is located by default at:

Director 8.5 - C:\Program Files\Macromedia\Director 8.5\xtrainfo.txt

Director MX - C:\Program Files\Macromedia\Director MX\xtrainfo.txt

Double click the file to open it in notepad, or alternatively edit with any other text editor. You need to add the following line to the end of the file:

```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://download.tabuleiro.com/packages/TreeView/4/TreeView"]
```

You may want to customize this line in the future, to instruct Director to download TreeView Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The

Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button (Director MX).

TREEVIEW XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR 11

INSTALLING THE XTRA ON MAC OSX - Director 11

Double-click the installation .dmg file. This will mount a disk named "TreeView" on your desktop.

The first step is to copy the Universal binary version of the Xtra, which will be used in the authoring environment and also when creating Mac OSX projectors, for both Intel and PPC machines. This file is located in the install disk image, at:

TreeView/Mac Universal/TreeView.xtra

This file needs to be copied to the Director 11 Xtras folder. The final pathname for the Xtra in a default installation of Director 11 will be:

OSX Volume Name/Applications/Adobe Director 11/Configuration/Xtras/TreeView.xtra

Windows projectors can also be created directly on Director 11 running on Mac OSX after installation of the Windows version of the Xtra. It is located on the install disk, at:

TreeView/Windows/TreeView.x32

Copy this file to the Cross Platform resources directory in Director 11, so that it will be available at:

OSX Volume Name/Applications/Adobe Director 11/Configuration/Cross Platform Resources/Windows/Xtras/TreeView.x32

Finally, you need to edit the xtrainfo.txt file to include information about TreeView. This information is used by the Shockwave and cross-platform publishing features in Director 11, to locate the files needed when assembling the Windows version of your projector. The xtrainfo.txt file is located by default at:

```
OSX Volume Name/Applications/Adobe Director 11/Configuration/xtrainfo.txt
```

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://download.tabuleiro.com/packages/TreeView/4/TreeView"]
```

You may want to customize this line in the future, to instruct Director to download TreeView Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.

TREEVIEW XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR MX 2004

INSTALLING THE XTRA ON MAC OSX - Director MX 2004

Double-click the installation .dmg file. This will mount a disk named "TreeView" on your desktop.

The first step is to copy the OSX version of the Xtra, which will be used in the authoring environment and also when creating OSX projectors. This file is located in the install disk image, at:

TreeView/Mac Carbon/TreeView

This file needs to be copied to the Director MX 2004 Xtras folder. The final pathname for the OSX Xtra in a default installation of Director MX will be:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Xtras/TreeView

Director MX 2004 running on Mac OSX can also be used to create Classic projectors, for Mac OS versions 8 and 9. In order to enable this feature you need to copy the Classic version of TreeView to the correct location in your Director MX installation. First locate the Classic version of TreeView in the install disk:

TreeView/Mac Classic/TreeView

This file needs to be copied to the following location in the Director MX 2004 folder, to be used for cross-platform publishing. Copy it to:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Cross Platform Resources/Classic MacOS/Xtras/TreeView

Windows projectors can also be created directly on Director MX 2004 running on Mac OSX after installation of the Windows version of the Xtra. It is located on the install disk, at:

TreeView/Windows/TreeView.x32

Copy this file to the Cross Platform resources directory in Director MX 2004, so that it will be available at:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/Cross Platform Resources/Windows/Xtras/TreeView.x32

Finally, you need to edit the xtrainfo.txt file to include information about TreeView. This information is used by the Shockwave and cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the Classic MacOS and Windows versions of your projector. The xtrainfo.txt file is located by default at:

OSX Volume Name/Applications/Macromedia Director MX 2004/Configuration/xtrainfo.txt

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://download.tabuleiro.com/packages/TreeView/4/TreeView"]
```

You may want to customize this line in the future, to instruct Director to download TreeView Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The

Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.

TREEVIEW XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR MX

INSTALLING THE XTRA ON MAC OSX - Director MX

Double-click the installation .dmg file. This will mount a disk named "TreeView" on your desktop.

The first step is to copy the OSX version of the Xtra, which will be used in the authoring environment and also when creating OSX projectors. This file is located in the install disk image, at:

TreeView/Mac Carbon/TreeView

This file needs to be copied to the Director MX Xtras folder. The final pathname for the OSX Xtra in a default installation of Director MX will be:

OSX Volume Name/Applications/Macromedia Director MX/Xtras/TreeView

Director MX running on Mac OSX can also be used to create Classic projectors, for Mac OS versions 8 and 9. In order to enable this feature you need to copy the Classic version of TreeView to the correct location in your Director MX installation. First locate the Classic version of TreeView in the install disk:

TreeView Folder/Mac Classic/TreeView

This file needs to be copied to the following location in the Director MX folder:

OSX Volume Name/Applications/Macromedia Director MX/Classic MacOS/Xtras/TreeView

Finally, you need to edit the xtrainfo.txt file to include information about TreeView. This information is used by the Shockwave and cross-platform publishing features in Director MX 2004, to locate the files needed when assembling the Classic MacOS version of your projector. The xtrainfo.txt file is located by default at:

```
OSX Volume Name/Applications/Macromedia Director MX/xtrainfo.txt
```

Double click the file to open it in TextEdit, or alternatively edit with another text editor. Make sure to save the file in plain text format, though. You need to add the following line to the end of the file:

```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://download.tabuleiro.com/packages/TreeView/4/TreeView"]
```

You may want to customize this line in the future, to instruct Director to download TreeView Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window. The Xtra functions will also be listed when you click the message window Scripting Xtras button.

TREEVIEW XTRA HELP: INSTALLATION: MACINTOSH - DIRECTOR 8.5

INSTALLING THE XTRA ON MAC OS 8 AND 9 - Director 8.5

Running under OSX, double-click the installation .dmg file. This will mount a disk named "TreeView" on your desktop. To install the Xtra just copy the file "TreeView" from the Mac Classic folder to the Xtras folder of your Director 8.5 installation. The final pathname for the Xtra will be for example:

Macintosh HD:OS9 Applications:Macromedia Director 8.5:Xtras:TreeView

Finally, you need to edit the xtrainfo.txt file to include information about TreeView. This information is used by the Shockwave publishing features in Director. The xtrainfo.txt file is located in the directory where Director 8.5 was installed, for example at:

Macintosh HD:OS9 Applications:Macromedia Director 8.5:xtrainfo.txt

Double click the file to open it in SimpleText, or another editor capable of saving plain text files. You need to add the following line to the end of the file:

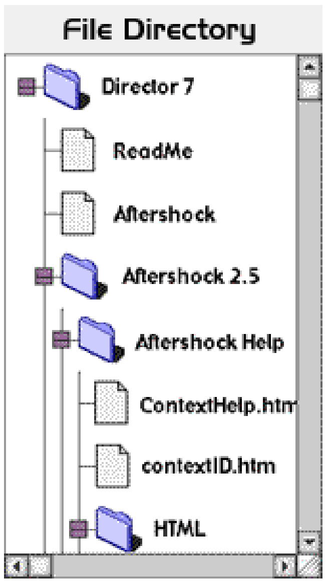
```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://download.tabuleiro.com/packages/TreeView/4/TreeView"]
```

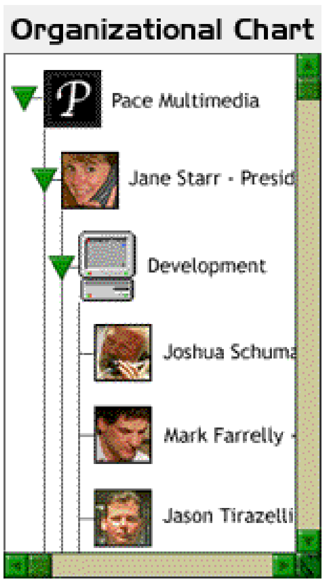
You may want to customize this line in the future, to instruct Director to download TreeView Shockwave packages from the same server that hosts your Shockwave applications. This is covered in more detail at the Using the Xtra in Shockwave section of the documentation. Restart Director for the changes to take effect. The Xtra should be listed when you issue the command "put the xtralist" in the message window.

TREEVIEW XTRA HELP: WHAT TREEVIEW XTRA DOES

TreeView displays information in a hierarchical tree format similar to Windows Explorer or the Macintosh's Finder. You can customize the tree, by choosing the text font, icons, connector types, spacing and other properties. Since you structure the tree branches and supply the data for the node names, you have great flexibility to present a variety of different kinds of hierarchical data. Each node can report back both rollovers and mouse clicks, and automatically handles highlighting if you supply a highlight image.

Any type of hierarchical data that you want the user to browse and select can be presented with TreeView Xtra. The graphics below show a file browser and an interactive organizational chart.





Other applications include:

- Multi-tiered result records from a database query. Presenting the results in tree format rather than as a flat report enables the user to drill down quickly to the areas the user is interested in. For instance sales records could be presented organized by region, then salesperson within region, then product within salesperson.

- Navigational map of CD or web site.

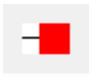
- Displaying the structure of an XML document

- Collapsible outline for navigating an electronic book

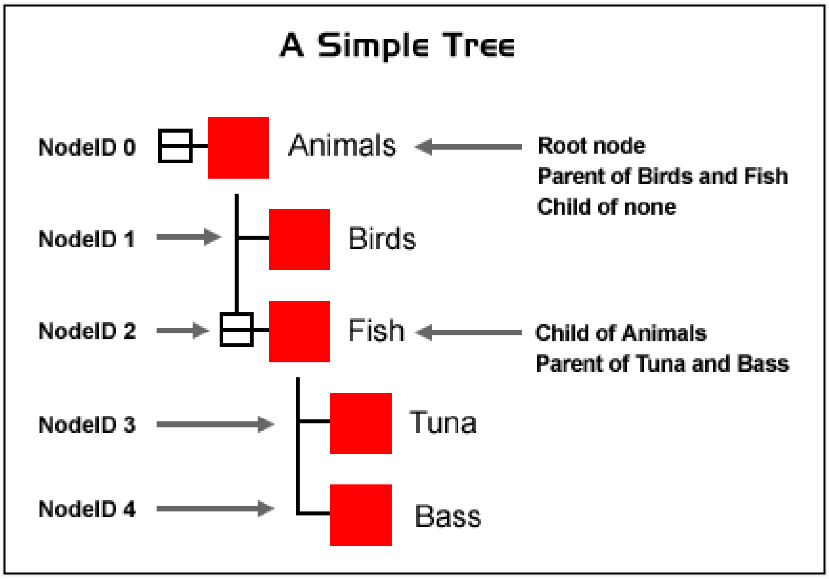
TREEVIEW XTRA HELP: GETTING STARTED

TreeView is an Asset Xtra. Unlike Scripting Xtras, Asset Xtras can be manipulated using the score and cast windows, and their properties can be adjusted through scripting, just like Director's built-in media types. You create a new TreeView cast member by highlighting an empty cast slot and choosing Insert -> Tabuleiro Xtras -> TreeView from Director's menu. If you drag the newly created cast member onto the stage you will see a TreeView tree sprite with one default node called the root node. Resize the sprite to the size you want the viewable area of the tree to be. Create an approximately 200 x 200 pixel sprite to practice.

Initial appearance of a TreeView sprite:



In TreeView terms, nodes are the branches of the tree. A parent node contains other nodes. A child node is contained by a parent node. A node can be the child of a higher-up parent node, and have children of its own at the same time. The only node that is not a child node of some other node is the root node because it is at the top of the tree.You fill a tree by adding nodes to it, one at a time and specifying their properties, such as the node text and icon. When a node is created it is assigned a nodeID, which is a unique number.



This is the code that created the tree above, written as a behavior attached to the TreeView sprite:

```
on beginSprite me

makeSmallTree()

end

on makeSmallTree

root = sprite(me.spritenum).getRootNodeXTV( )

sprite(me.spritenum).setNameXTV(root,"Animals")

birdsNodeID = sprite(me.spritenum).addChildXTV( root, 1, 0)

sprite(me.spritenum).setNameXTV(birdsNodeID,"Birds")

fishNodeID = sprite(me.spritenum).addChildXTV( root, 1, 0)

sprite(me.spritenum).setNameXTV(fishNodeID,"Fish")

tunaNodeID = sprite(me.spritenum).addChildXTV( fishNodeID, 1, 0)

sprite(me.spritenum).setNameXTV(tunaNodeID,"Tuna")

bassNodeID = sprite(me.spritenum).addChildXTV( fishNodeID, 1, 0)

sprite(me.spritenum).setNameXTV(bassNodeID,"Bass")

end
```

The setNameXTV method sets the text of the node. The addChildXTV method creates a new node. Note that by specifying the fishNodeID as the parent in the last two addChildXTV lines, new nodes were created under node "Fish" instead of directly under the root node "Animals".

The simple tree above uses the default red box node icon. To assign graphics from your Director cast to nodes, see the Icon Section. The background of a TreeView sprite is transparent, and the sprite has no bounding outline, which allows you to place graphics behind it and to frame it any way you like.You can also add scrollbars, configure the tree layout and change the font.

When the user rolls over or clicks on a node's icon, the action generates a callback message. You trap and act on the callback message in your code by creating a movie script with a designated name to handle the type of message you want to trap. For instance if you wanted to be informed of the user's single clicks on tree nodes you would create the following handler in a movie script:

```
on treeViewSingleClick refcon, nodeid, whichPart, spriteChannel

-- refcon: internal treeview id of tree sprite that was clicked

-- id: nodeID of the node that was clicked

-- whichPart: area of node that was clicked

-- 1 = icon area, 2 = text area

-- spriteChannel: sprite number of tree sprite

--

-- Your code below

--

put sprite(spriteChannel).getNameXTV(nodeid) into field "choice"

end
```

Note: the refcon is an internal TreeView number assigned to the TreeView sprite. It is not the same as the sprite number. You can use it to distinguish between different trees assigned to the same sprite channel in different frames of the movie:

```
sprite(3).setRefconXTV (5)
```

TREEVIEW BEHAVIORS

The TreeView demonstration movie contains drag and drop behaviors that can build a tree from a hierarchy specified in a field, build a tree from a folder on the user's hard drive or build thumbnails from a folder of images. Using the behaviors does not require any Lingo scripting. Simply drag one of the behaviors to a TreeView sprite and fill out the parameters. Each behavior contains full documentation at the top of the script that explains how to fill out any parameter you may not understand.

The TreeView behaviors were intended for use by Lingo novices. In order to be as general-purpose as possible they offer the ability to customize every TreeView property that exists. The trade-off for this degree of flexibililty is slower performance when creating the tree, since all possible properties must be set for each tree node.

If you are an experienced Lingo programmer, and you are interested in displaying the tree initially as quickly as possible, you should use the makeSmallTree example as a starting point and keep the following perfomance tips in mind:

- Lingo repeat loops lock user interaction out. If you are creating a very large tree (1000's of nodes) you may want to temporarily increase the frame rate and use an exitFrame handler that adds one node per frame, rather than a repeat loop.This method allows the user to continue working while the tree is being generated. If you do not want the user to interact with the tree until it has finished building, keep the tree sprite off stage (negative locV) until it has completely built, and then move the sprite onto the stage.

- If you do choose to use a repeat loop, remove any Lingo lines from the loop that do not have to execute every time a node is created. Avoid doing any text manipulation if possible such as altering strings, or the text inside fields or text members. These are the slowest operations in Lingo and will bog down the loop.

- Set the least number of properties possible at the node level. Take advantage of treewide commands like getGlobalIconIDXTV and setAllNodesIconIDXTV to set properties that will be the same for all nodes. If you are going to take the default for a property, for instance if you are going to use the system font for every tree node, do not set that property when creating the node.

- Do not set a variable to the result of addChildXTV unless you need the variable farther down in the handler to add properties or children to the node. The functions getNumChildrenXTV , getNthChildXTV , findChildXTV , getParentXTV and getNameXTV will help you find any node by name later on. In particular, do not create a global variable for each nodeID of the tree. This is unnecessary and may use up all of your variable space.

REGISTRATION

The unregistered version displays a registraton warning, and is fully functional for evaluation purposes.

If you have purchased TreeView Xtra you received a serial number that can be used to remove the warning. Make the following registration call once the first time that each TreeView sprite appears in the score. Make sure you do it before using any other TreeView commands on the sprite. Put the code in a sprite script on theTreeView sprite on stage::

```
on beginSprite me

sprite(me.spritenum).setMagicXTV("48dkd2929")

-- your registration number is the second parameter

end
```

TREEVIEW XTRA HELP: METHODS AT A GLANCE

Method and arguments	Purpose
sprite(spriteReference).setMagicXTV (registrationCodeString)	Prevents the demo dialog from coming up after purchase.

Node Methods

Add/Delete Node

sprite(spriteReference).getRootNodeXTV ( )	Returns the node ID of the top node on the tree
sprite(spriteReference).addChildXTV (parentNID, addWhere, preceedingNID)	Adds a new node to the tree as a child of the specified parent node.
sprite(spriteReference).removeChildXTV (parentNID, childNID)	Removes the specified node ID from the tree as well as any of its children.
sprite(spriteReference).removeAllChildrenXTV (parentNID)	Removes all of the children of the specified node as well as their children.
sprite(spriteReference).getNumChildrenXTV (nodeID)	Returns the number of top-level children of that node.
sprite(spriteReference).getNthChildXTV (parentNID, childPosition)	Returns the nodeID of the child node in the nth position.
sprite(spriteReference).findChildXTV (parentNID, childNID)	Returns the node position, not nodeID, of the child node specified.
sprite(spriteReference).getParentXTV (childNID)	Returns the node ID of the parent node of the specified child node

Node Properties

sprite(spriteReference).getNodeLocationXTV (nodeID)	Returns the coordinates of the top, left corner of the specified node's bounding box.
sprite(spriteReference).setNameXTV (nodeID, nodeName)	Sets the text label displayed for the specified node.
sprite(spriteReference).getNameXTV (nodeID)	Returns the name of the node specified by nodeID.
sprite(spriteReference).setIconIDXTV (nodeID, iconType, iconID)	Sets the icon image used for a particular icon type for an individual node.
sprite(spriteReference).getIconIDXTV (nodeID, iconType)	Returns the icon ID for the specified icon type of the node specified.
sprite(spriteReference).setFontIDXTV (nodeID, fontID)	Sets the font that an individual node's text will display in.
sprite(spriteReference).getFontIDXTV (nodeID)	Returns the fontID from the internal font list currently set for the node.
sprite(spriteReference).setFontColorXTV (nodeID, textRed, textGreen, textBlue)	Sets the text color for the specified node.
sprite(spriteReference).getFontColorXTV (nodeID)	Returns a string containing the current RGB value set for the node's text color.
sprite(spriteReference).setExpandedXTV (nodeID, expandedFlag)	Expands a node to show the node's top-level children.
sprite(spriteReference).getExpandedXTV (nodeID)	Returns the current expand state of the node.
sprite(spriteReference).setUserNumXTV (nodeID, customNumber)	Stores a custom number supplied by the Lingo programmer with the node.
sprite(spriteReference).getUserNumXTV (nodeID)	Returns the custom number previously stored with the node using setUserNumXTV
sprite(spriteReference).setUserStringXTV (nodeID, stringNum, customString)	Links a string defined by the Lingo programmer to a node.

sprite(spriteReference).getUserStringXTV (nodeID, stringNum)	Returns the specified custom data string stored with a node using setUserStringXTV.
sprite(spriteReference).setMouseStatusRequestXTV (nodeID, singleClickFlag, doubleClickFlag, mouseEnterFlag, mouseOverFlag, mouseDownFlag)	Determines per node which user actions will be reported to the callback handlers.
sprite(spriteReference).getMouseStatusRequestXTV (nodeID)	Returns a string with 5 flags, either 1 or 0, separated by spaces.
sprite(spriteReference).setAllNodesMouseStatusRequestXTV (singleClick, doubleClick, mouseEnter, mouseOver, mouseDown)	Resets for all existing nodes which user actions will be reported to the callback handlers.

Tree Methods

Icon Properties

sprite(spriteReference).getNumIconsXTV ( )	Returns the total number of icons in the tree's icon list.
sprite(spriteReference).addIconXTV (castlibNum, memberNum, paletteNum, pixelWidth, pixelHeight)	Adds a bitmap cast member to the tree's icon list.
sprite(spriteReference).getIconXTV (iconID)	Retuns information for the stored icon
sprite(spriteReference).setIconXTV (iconID, castlibNum, memberNum, paletteNum, pixelWidth, pixelHeight)	Resets the properties of an existing icon.
sprite(spriteReference).getMaxIconSizeXTV (iconType)	Returns the maximum height and width of the specified icon type.
sprite(spriteReference).setMaxIconSizeXTV (iconType, pixelWidth, pixelHeight)	Limits the height and width of the specified icon type.
sprite(spriteReference).setGlobalIconIDXTV (iconType, iconID)	Sets the icon graphic used throughout the tree for the expander icon type if the property has not yet been set

	individually for the node.
sprite(spriteReference).getGlobalIconIDXTV (iconType)	Returns the global icon ID set for the specified icon type.
sprite(spriteReference).setAllNodesIconIDXTV (iconType, iconID)	Resets the icon for the specified icon type for existing nodes.
sprite(spriteReference).resetScrollbarsXTV(0)	Scrolls the tree back up to the top/left.

Connector Line and Spacing Properties

sprite(spriteReference).setLineModeXTV (lineType)	Controls the type of line used for the connecting lines between nodes.
sprite(spriteReference).getLineModeXTV ( )	Returns the current line type used for the connecting lines between nodes.
sprite(spriteReference).setMainColorsXTV (lineRed, lineGreen, lineBlue)	Sets the color of the connecting lines between nodes.
sprite(spriteReference).getContentSizeXTV ( )	Returns a string containing the width and height in pixels of a tree's content area.
sprite(spriteReference).setContentOffsetXTV (horizontalPixels, verticalPixels)	Moves content area within TreeView sprite.
sprite(spriteReference).setSpacingXTV (indent, iconSideMargin, iconTopMargin, expanderSideMargin, expanderTopMargin)	Adjusts the spacing between tree icons.
sprite(spriteReference).getSpacingXTV ( )	Returns a string containing the tree's current spacing settings separated by spaces.

Scroll Bar Properties

sprite(spriteReference).setScrollbarUsageXTV (onOrOff)	Shows or hides the scrollbars for a TreeView sprite.

sprite(spriteReference).setScrollbarDataXTV (castlibNum, startingMemberNum, 0)	Sets the images to use for the various parts of the vertical and horizontal scrollbar.
sprite(spriteReference).setScrollbarColorsXTV (inactiveRed, inactiveGreen, inactiveBlue, frameRed, frameGreen, frameBlue,activeRed, activeGreen,activeBlue)	Sets the color used to draw the frame outline of the scrollbars and the color that fills the interior of the scroll bar tracks.
sprite(spriteReference).resetScrollbarsXTV(0)	Scrolls the tree back up to top/left.

Text Properties

sprite(spriteReference).addFontXTV (fontName, fontSize, fontStyle)	Adds a font to TreeView's internal font list.
sprite(spriteReference).setFontXTV (fontID, fontName, fontSize, fontStyle)	Modifies the properties of a font in the font list.
sprite(spriteReference).getFontXTV (fontID)	Returns a string containing font properties set for a font in the font list.
sprite(spriteReference).getNumFontsXTV ( )	Returns the number of fonts currently in TreeView's font list.
sprite(spriteReference).setTextHiliteColorXTV (textRed, textGreen, textBlue)	Sets the highlight color that displays over node text when the user clicks on a node.

Miscellaneous Properties

sprite(spriteReference).getVersionXTV ( )	Returns the version string for TreeView.
sprite(spriteReference).setRefconXTV (newID)	Sets the id number that is passed to TreeView callback handlers to identify the tree that the user clicked.
sprite(spriteReference).getRefconXTV ( )	Returns the current refcon number of the TreeView sprite.

sprite(spriteReference).redrawXTV ( )	Updates the TreeView sprite display after visual tree properties have been changed.

TREEVIEW XTRA HELP: METHODS DOCUMENTATION

sprite(spriteReference).setMagicXTV(registrationCodeString) - where spriteReference is the sprite channel containing the TreeView sprite and registrationCodeString is the serial number assigned at purchase. Always returns -1.

TreeView Xtra demo version will display a trial-version alert the first time you make a call to it, and will only allow you to create 20 nodes. The full version of TreeView Xtra requires a serial number. If you have purchased the TreeView Xtra you received a registration number along with the full version of TreeView Xtra. Replace the demo version of TreeView Xtra with the full version in your Xtras folder. Before sending any other commands to a TreeView sprite, first call the register method like so:

sprite(3).setMagicXTV("48dkd2929")

-- your registration number is the second parameter

Each TreeView sprite in a movie must be registered before it will respond to TreeView calls.

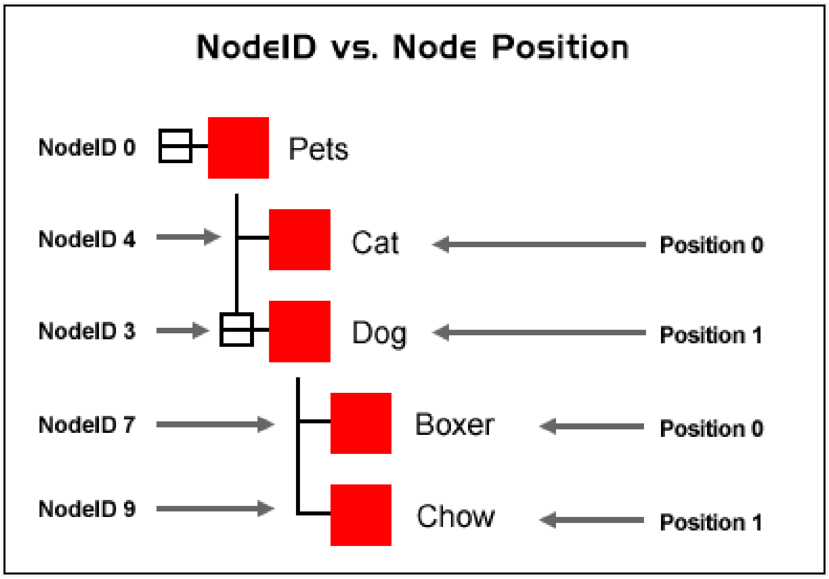
NOTE: A beginsprite handler executes BEFORE an exitframe handler. Make sure that setMagicXTV is in a handler that will execute before any other treeview calls. If setMagicXTV is in a frame script, do not put TreeView calls in a beginSprite handler if the sprite is on the same frame.

Example:

sprite(3).setMagicXTV("48dkd2929")

NODE ADD/DELETE METHODS

The following methods add or remove nodes from a tree or return information that helps determine which node you want to add or delete. There are two ways of identifying nodes - by nodeID and by position relative to other nodes of the same parent. NodeID is a unique number assigned to the node at node creation. Deleting or adding a node does not change the node ID's of other nodes. Even though node ID's are assigned sequentially at creation, once nodes have been deleted and more have been added, the node ID will no longer correspond to its position on the tree. You should not use nodeID as any indication of where the node is in relation to other nodes of the same parent.



Conversely, the node position property of a node does depend on what other nodes with the same parent currently exist at the same level and does change if nodes are added to or deleted from the parent. Node position starts at 0 for the top node and increments by 1. Use node position to cycle through all of a node's children. The function getNthChild returns nodeID's when passed node position.

sprite(spriteReference).getRootNodeXTV( ) - where spriteReference is the sprite channel containing the TreeView sprite. Returns the node ID of the top node on the tree. Returns the nodeID of the root node of the tree. Every TreeView tree starts off with one root node, to which other nodes can be added. The root node ID is usually 0. The root node cannot be deleted.

Example:

```
set root = sprite(2).getRootNodeXTV()

-- set name of root and hence name of tree

sprite(2).setNameXTV(root,"Accounting Records 1999")
```

sprite(spriteReference).addChildXTV(parentNID,addWhere,preceedingNID) - where spriteReference is the sprite channel containing the TreeView sprite; parentNID is the integer node ID of the parent node; addWhere is the integer position to add the child to among existing child nodes (0 => add the new node as the first child, 1 => add the new node as the last child, 2 => add the new node after the child indicated by preceedingNID); and preceedingNID is the integer node ID of th node to add the new node after (Only used if addWhere argument is 2. Otherwise pass 0 in this argument). Returns the nodeID of the new node or -1 if the operation is unsucessful.

Adds a new node to the tree as a child of the specified parent node. The new node can be placed before or after all of the existing children or after the child node specified in preceedingNID. The nodeID assigned to a newly created node is unique for the life of the tree sprite. For example, if you create a tree with 100 nodes initially, delete all of the nodes, and add 100 more, the nodeID for the first node in the second batch of nodes will be 101.

Example:

root = sprite(2).getRootNodeXTV()

-- adds new child after any existing children of root

newNodeID = sprite(2).addChildXTV(root, 1, 0)

sprite(spriteReference).removeChildXTV(parentNID,childNID) - where spriteReference is the sprite channel containing the TreeView sprite, parentNID is the integer node ID of the parent node and childNID is the integer nodeID of the child of the parent node specified. Returns 0 if successful, or negative number if the operation unsuccessful. Removes the specified node ID from the tree as well as any of its children.

Example:

on killByName treeSprite,parentID,theName

-- Deletes any node of specified parentID that has

-- the specified name

--

-- EX: killbyname(sprite 1,0,"Help")

--

set numChildren = sprite(treeSprite).getNumChildrenXTV(parentID)

if numChildren > 0 then

-- since index starts at 0, delete 1 to get last one

set lastChild = numChildren - 1

repeat with x = 0 to lastChild

```
-- get nodeID at each position, starting at 0

set nodeID = sprite(treeSprite).getNthChildXTV(parentID,x)

-- if node's name is the one we want, delete it

if sprite(treeSprite).getNameXTV(nodeID) = theName then

sprite(treeSprite).removeChildXTV(parentID,nodeID)

exit repeat

end if

end repeat

end if

end
```

sprite(spriteReference).removeAllChildrenXTV(parentNID) - where spriteReference is the sprite channel containing the TreeView sprite and parentNID is the integer node ID of the parent node. Returns 0 if successful or a negative number if not. Removes all of the children of the specified node as well as their children. If the root node is used, this clears tree.

Example:

```
root = sprite(2).getRootNodeXTV()

-- wipes out all of the nodes when applied to root

err = sprite(2).removeAllChildrenXTV(root)
```

sprite(spriteReference).getNumChildrenXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite and nodeID is the integer nodeID of the node to count children of. Returns the number of children or -1 if not successful. Returns the number of top-level children of that node. Does not count any children of the top-level nodes. Use the example handler to count all children including children of children.

Example:

```
on countNodes treeSprite, startNode, startCount
```

```
-- treeSprite: sprite number of tree

-- startNode: nodeID to start counting at

-- startCount: always pass 0

--

-- Returns the total number of nodes contained by specified node

-- including "grandchildren" (contained by children)

-- If passed root node, returns the total number of nodes in tree

-- but does not count root node. Add 1 to result to count root node

-- Always pass 0 for startCount

--

-- EX: treeCount = countNodes(1,0,0)

-- treeCount = treeCount + 1

--

nodeID = startNode

curcount = startCount

numChildren = sprite(treeSprite).getNumChildrenXTV(nodeID)

curcount = curcount + numChildren

lastChild = numChildren - 1

repeat with pos = 0 to lastChild

childID = sprite(treeSprite).getNthChildXTV(nodeID,pos)

curCount = countNodes(treeSprite,childID,curcount)

end repeat

return curcount

end
```

sprite(spriteReference).getNthChildXTV(parentNID,childPosition) - where spriteReference is the sprite channel containing the TreeView sprite, parentNID is the integer nodeID of the parent node and childPosition is the integer position relative to other child nodes, not nodeID. Returns the node ID of the node in the specified position or -1 if not successful. Returns the nodeID of the child node in the nth position. Positions are numbered from 0, top node, and increment by 1.

Example:

```
on showNames treeSprite, nodeID

-- Prints out the child position and name for the top-level

-- children of a given nodeID

--

-- EX: shownames(1,0)

--

-- Example output

--

-- "0: Acrobat Reader 4.0 Install Log"

-- "1: Acrobat™ Reader 4.0"

-- "2: Help:"

-- "3: Plug-Ins:"

--

numChildren = sprite(treeSprite).getNumChildrenXTV(nodeID)

lastChild = numChildren - 1

repeat with pos = 0 to lastChild

childID = sprite(treeSprite).getNthChildXTV(nodeID,pos)

put pos & ": " & sprite(treeSprite).getNameXTV(childID)

end repeat

end
```

sprite(spriteReference).findChildXTV(parentNID,childNID) - where spriteReference is the sprite channel containing the TreeView sprite, parentNID is the integer node ID of the parent node and childNID is the integer nodeID of the child of the parent node specified. Returns the node position, not nodeID, of the child node specified or -1 if unsuccessful. This is the complimentary function to getNthChildXTV, which returns nodeID when passed position.

Example:

```
on treeViewSingleClick refcon, nodeID, whichPart,spriteChan

-- Callback handler coded to detect when user

-- clicked the first node in any containing node

-- Assumes that tree sprite's refcon property has

-- been set to its sprite number

--

parentID = sprite(refcon).getParentXTV(nodeID)

parentName = sprite(refcon).getNameXTV(parentID)

pos = sprite(refcon).findChildXTV(parentID,nodeID)

if pos = 0 then alert "You clicked the first node in " & parentName

end
```

sprite(spriteReference).getParentXTV(childNID) - where spriteReference is the sprite channel containing the TreeView sprite and childNID is the integer nodeID of the child of the parent node specified. Returns the nodeID of the parent of the specified node or -1 if the child does not have a parent. Will only return -1 for the root node, since all other nodes must have parents.

Example:

```
on treeViewSingleClick refcon, nodeID, whichPart,spriteChan

-- Callback handler coded to report name of
-- containing node of any node clicked

-- Assumes that tree sprite's refcon property has

-- been set to its sprite number

--

parentID = sprite(refcon).getParentXTV(nodeID)

if parentID > -1 then

parentName = sprite(refcon).getNameXTV(parentID)

alert "You clicked a node in " & parentName

else

alert "This node has no parent"

end if

end
```

NODE PROPERTIES

sprite(spriteReference).getNodeLocationXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite and nodeNID is the integer node ID of the node. Returns a string containing coordinates for the node in format "h v" or "ERRORCODE=-50" if the node does not exist.

Returned string contains horizontal and vertical coordinates of the top, left point of the node's bounding box. A node's bounding box is the smallest rectangle that encloses the node, text and the expander area and the vertical spacing specified between nodes. A node's bounding box always includes room for an expander icon even if the node is not a parent node and does not have an expander.

The coordinates returned are relative to the TreeView sprite's top/left corner. In order to get stage coordinates you must add the TreeView Sprite's value for it's top/left to the value's returned. See example handler.

Example:

```
on showNode treeSprite,nodeID
```

```
-- positions sprite over nodeID passed in

pointerGraphicSprite = 80

locstring = sprite(treesprite).getNodeLocationXTV( nodeID)

-- add the coords returned to the treeview sprite's coords to get coords for positioning

-- a sprite over the node

sph = sprite(treesprite).rect[1]

spv = sprite(treesprite).rect[2]

sprite(pointerGraphicSprite).loc = point(value(word 1 of locstring) + sph, value(word 2 of locstring) + spv)

end
```

sprite(spriteReference).setNameXTV(nodeID,nodeName) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node and nodeName is the string node text. Returns 0 if successful or a negative number if unsuccessful. Sets the text label displayed for the specified node.

Example:

```
root = getRootNodeXTV(sprite 2)

-- set name of root and hence name of tree

sprite(2).setNameXTV(root,"Accounting Records 1999")
```

sprite(spriteReference).getNameXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite and nodeNID is the integer node ID of the node. Returns a string containing the node name or "". Returns the name of the node specified by nodeID. Returns an empty string if the node is unnamed or the node is not found.

Example:

```
on treeViewSingleClick refcon, nodeID, whichPart,spriteChan

-- Assumes that tree sprite's refcon property has
-- been set to its sprite number

--

alert "You clicked node " & sprite(refcon).getNameXTV(nodeID)

end
```

sprite(spriteReference).setIconIDXTV(nodeID,iconType,iconID) - where:

spriteReference is the sprite channel containing the TreeView sprite,

nodeNID is the integer node ID of the node,

iconType is the integer specifying which icon type to set for the node. Must be one of the following values:

0: normal icon

1: highlighted icon

2: plus expander

3: minus expander

iconID is the integer icon ID from the internal icon list.

Returns 0 if successful or a negative number if not.

Sets the icon image used for a particular icon type for an individual node. The icon ID specified refers to an icon in the sprite's internal icon list, which contains entries that point to bitmap cast members. You cannot reference cast members directly.

Depending on the size of your node icons and your layout choices, the last icon at the root level of a tree can appear partially under the scrollbar when the user scrolls all the way to the bottom of the tree. If this happens and you don't want to change your layout, you can fix it by adding an icon to the icon list with a 0-pixel height and width, adding a child node to the root, and assigning the new node the blank icon with setIconIDXTV. The new blank node will not be visible to the user or accept user clicks, but it will create a space at the bottom of the tree so that the last real node displays fully.

Example:

```
-- Sets the highlighted icon for node 3

sprite(2).setIconIDXTV(3,1,8)
```

sprite(spriteReference).getIconIDXTV(nodeID,iconType) - where:

spriteReference is the sprite channel containing the TreeView sprite,

nodeNID is the integer node ID of the node,

iconType is the integer specifying which icon type to examine for the node. Must be one of the following values:

0: normal icon

1: highlighted icon

2: plus expander

3: minus expander

Returns the icon ID from the internal icon list of the icon type specified or -1 if the icon is not set.

Returns the icon ID for the specified icon type of the node specified. Each node is created with normal and highlighted set to the default (0) and plus and minus unset (-1)

Example:

```
on treeViewSingleClick refcon, nodeID, whichPart,spriteChan

-- Uses the name of the cast member associated

-- with a node's icon to determine if a folder

-- or document has been clicked

--

-- Assumes that tree sprite's refcon property has

-- been set to its sprite number

--

iconID = sprite(refcon).getIconIDXTV(nodeID,0)

iconInfo = sprite(refcon).getIconXTV(iconID)
```

```
cast = value(word 1 of iconinfo)

mem = value(word 2 of iconinfo)

nam = the name of member mem of castlib cast

if nam contains "folder" then

alert "You clicked a folder"

else

alert "You clicked a document"

end if

end
```

sprite(spriteReference).setFontIDXTV(nodeID,fontID) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node and fontID is the integer font ID from the internal font list. Returns 0 if successful or a negative number if not. Sets the font that an individual node's text will display in. Use setFontXTV to set the font tree-wide for all nodes.

Example:

```
-- set node 3's font to fontID 4

err = sprite(1).setFontIDXTV(3,4)
```

sprite(spriteReference).getFontIDXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node. Returns the font ID of the specified node or -1 if unsuccessful. Returns the fontID from the internal font list currently set for the node.

Example:

```
fontID = sprite(2).getFontIDXTV(3)
```

sprite(spriteReference).setFontColorXTV(nodeID,textRed,textGreen,textBlue) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node, textRed is the integer RGB red value, textGreen is the integer RGB green value, textBlue is the integer RGB blue value. Returns 0 if successful or a negative number if not. Sets the text color for the specified node. Default is black.

Example:

err = sprite(1).setFontColorXTV(3,65535,20000,20000)

sprite(spriteReference).getFontColorXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite and nodeNID is the integer node ID of the node. Returns a string containing the current RGB value set for the node's text color. If there was an error, returns a string where the value of the first word of the string is > 65535. The string looks like this:

"65535 20000 20000"

If there was an error, the string looks like this:

"33779664 33779736 24024528"

Example:

colorString = sprite(1).getFontColorXTV(23)

if value(word 1 of colorString) > 65535 then

alert "Error getting node's color value"

end if

sprite(spriteReference).setExpandedXTV(nodeID,expandedFlag) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node and expandedFlag is 1 for TRUE, 0 for FALSE. Returns 0 if successful or a negative number if not. Expands a node to show the node's top-level children. Equivalent to the user clicking on the expander icon next to the node.

Example:

-- collapse the root node, which collapses

-- the entire tree

sprite(1).setExpandedXTV(0,0)

sprite(spriteReference).getExpandedXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite and nodeNID is the integer node ID of the node. Returns the current expand state of the node. Returns 1 if the node is currently expanded and 0 if it's not. Also returns 0 if passed a non-existent node.

Example:

expanded = sprite(1).getExpandedXTV(33)

sprite(spriteReference).setUserNumXTV(nodeID,customNumber) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node and customNumber is the integer number supplied by the programmer. Returns 0 if successful or a negative number if not.

Stores a custom number supplied by the Lingo programmer with the node. This is a very convenient way of associating data other than the node name with a particular node. Since TreeView's callback handlers pass the clicked nodeID, you car easily get both the node name (from getNameXTV) and the user number (from getUserNumXTV) For instance if the data was coming from a store catalog you might want to store the integer price of the item in cents with the item's node, rather than maintain a separate Lingo list for the information.

Example:

nodeID = sprite(1).addChildXTV(0,0,0)

sprite(1).setNameXTV(nodeID,"Allen wrench")

-- store $15.00 price with item (in cents)

sprite(1).setUserNumXTV(nodeID,1500)

sprite(spriteReference).getUserNumXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite and nodeNID is the integer node ID of the node. Returns the integer value of the custom number stored with the node, 0 if the node has no custom number, or -1 if the node does not exist. Returns the custom number previously stored with the node using setUserNumXTV.

Example:

customNum = sprite(1).getUserNumXTV(33)

sprite(spriteReference).setUserStringXTV(nodeID,stringNum,customString) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node, stringNum is an integer between 0 and 2 and customString is the string supplied by the Lingo programmer. Returns 0 if successful or a negative number if not.

Links a string defined by the Lingo programmer to a node. Like setUserNumXTV, this method enables you to link data other than node name to a specified node. You can store up to 3 strings of any length with a node. The argument stringNum specifies which storage location to use.

Example:

nodeID = sprite(1).addChildXTV(0,0,0)

sprite(1).setNameXTV(nodeID,"Allen wrench")

-- store part number

sprite(1).setUserStringXTV(nodeID,0,"KXJ0020")

-- store description

sprite(1).setUserStringXTV(nodeID,1,"Mighty fine wrench with gold handle")

sprite(spriteReference).getUserStringXTV(nodeID,stringNum) - where spriteReference is the sprite channel containing the TreeView sprite, nodeNID is the integer node ID of the node and stringNum is an integer between 0 and 2. Returns the custom string requested or "" if there is no custom string. Returns "" if the node or stringNum are invalid. Returns the specified custom data string stored with a node using setUserStringXTV.

Example:

on treeViewSingleClick refcon, nodeID, whichPart,spriteChan

-- Gets data stored with the node and writes it

-- to a field when the user clicks on the node.

```
--

-- Assumes that tree sprite's refcon property has

-- been set to its sprite number

--

partDescription = sprite(refcon).getUserStringXTV(nodeID,2)

put partDescription into field "description"

end
```

sprite (spriteReference).setMouseStatusRequestXTV (nodeID,singleClickFlag,doubleClickFlag,mouseEnter/exitFlag,mouseMoveFlag,mouseDownFlag) - where:

spriteReference is the sprite channel containing the TreeView sprite;

nodeNID is the integer node ID of the node;

singleClickFlag is 1 for on, 0 for off;

doubleClickFlag is 1 for on, 0 for off;

mouseEnter/exitFlag is 1 for on, 0 for off;

mouseMoveFlag is 1 for on, 0 for off;

and mouseDownFlag is 1 for on, 0 for off.

Returns 0 if successful or a negative number if not.

Determines per node which user actions will be reported to the callback handlers. Pass 0 for a user action you want to ignore or 1 to send the action to the callback handler. Pass 0 for all arguments to ignore all user activity for a particular node. You must pass 0 for the last argument even though it isn't used. The default settings at node creation are single click and doubleclick on, mouseEnter off. To set this property for all tree nodes at once use setAllNodesMouseStatusRequestXTV.

Example:

```
-- turn single click, double click and mouseenter/exit on for node 33
```

sprite(1).setMouseStatusRequestXTV(33,1,1,1,0,0)

sprite(spriteReference).getMouseStatusRequestXTV(nodeID) - where spriteReference is the sprite channel containing the TreeView sprite and nodeNID is the integer node ID of the node. Returns a string containing the current user events reported by the node. Returns a string with 5 flags, either 1 or 0, separated by spaces, representing the current state of event detection for the specified node. See setMouseStatusRequestXTV for flag definitions.

Example:

-- node 3 has singleClick and doubleClick on only

put sprite(1).getMouseStatusRequestXTV(3)

-- "1 1 0 0 0"

sprite (spriteReference).setAllNodesMouseStatusRequestXTV (singleClick,doubleClick,mouseEnter/exitFlag,mouseMoveFlag,mouseDownFlag) - where:

spriteReference is the sprite channel containing the TreeView sprite;

singleClickFlag is 1 for on, 0 for off;

doubleClickFlag is 1 for on, 0 for off;

mouseEnter/exitFlag is 1 for on, 0 for off;

mouseMoveFlag is 1 for on, 0 for off;

mouseDownFlag is 1 for on, 0 for off.

Always returns 0.

Resets for all existing nodes which user actions will be reported to the callback handlers. Nodes created after this method has been used will still be created with default callback settings. Use setMouseStatusRequestXTV to set this property per individual node.

Example:

```
-- turn all event detection on

sprite(1).setAllNodesMouseStatusRequestXTV(1,1,1,1,1)
```

ICON PROPERTIES

You can replace the default TreeView red box node icon with your own custom graphic stored in a Director cast member. You can also replace the plus and minus icons used for expanding or contracting parent nodes, and you can replace the graphic elements of the scrollbars, such as the drag box and scroll arrows. Node graphics can be specified tree-wide for simplicity, or you can set the graphics for each node individually. Each node has 4 icon types associated with it.

0: normal icon

1: highlighted icon

2: plus expander

3: minus expander

TreeView automatically displays the highlighted icon specified when the user clicks on the node, and automatically displays the minus icon when the node is expanded or the plus icon when the node has not been expanded. If any of the icon types have not been set for a node, TreeView uses the default TreeView graphic for that icon type.

TreeView icon methods do not reference cast members directly. They look to an internal TreeView icon list to get icon information. You use the addIconXTV method to add a cast member graphic to the icon list to make it available for use by any of the other icon methods. As each icon is added to the icon list TreeView assigns it a unique icon ID, similar to the nodeID's of TreeView nodes. You must use icon ID's to refer to icon graphics in the icon list, not cast member numbers.

The iconlist stores a string with each icon ID that specifies its properties in the following format:

castlibNumber memberNumber memberPalette maximumWidth maximumHeight

castlibNumber: number of cast library bitmap graphic is in

memberNumber: member number in cast library of graphic

memberPalette: member number in cast library of graphic's palette

maximumWidth: width in pixels to scale graphic to if larger maximumHeight: height in pixels to scale graphic to if larger

This is what a typical return from the getIconXTV method looks like. Icon 3 in the icon list is member 6 of castlib 1, and it will scale down to 16 x 16 if the graphic cast member is larger.

```
put sprite(1).getIconXTV(3)

-- "1 6 32 16 16"
```

Icon ID 0 is always the TreeView default red box icon. It cannot be modified or deleted. A TreeView sprite will always return at least 1 from getNumIconsXTV even if no custom icons have been added, because icon 0 is present. Note that the default icon returns -1 for castlib and member in its item information.

```
put sprite(1).getIconXTV(0)

-- "-1 -1 32 16 16"
```

sprite(spriteReference).getNumIconsXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. Returns an integer representing the number of icon images in the tree's icon list. Returns the total number of icons in the tree's icon list. The internal numbering of the list starts at 0 so subtract 1 from this number to get the last internal icon index number.

Example:

```
on showicons spriteNum

-- Lists information for each loaded icon image

num = sprite(spriteNum).getNumIconsXTV( )

-- icon index numbers start at 0

num = num - 1

repeat with x = 0 to num

iconInfo = sprite(spriteNum).getIconXTV(x)

put x & ") " & iconInfo

end repeat

end
```

sprite(spriteReference).addIconXTV(castlibNum,memberNum,memberPalette,pixelWidth,pixelHeight) - where spriteReference is the sprite channel containing the TreeView sprite, castlibNum is the integer cast library number of the castlib containing the member to use for the new icon, memberNum is the integer member number of the bitmap member to use for the new icon, memberPalette is the member number in the cast library of the graphic's palette, pixelWidth is the integer width in pixels for the icon and pixelHeight is the integer height in pixels for the icon. Returns a positive integer iconID of the icon added or -1 if the icon could not be added.

Adds a bitmap cast member to the tree's icon list. Adjusting the height and width of the icon will scale it. You can use this to make a thumbnail icon from a fullsize image. If you attempt to add the same member twice the ID of the existing icon is returned and no action is taken. **Notice**: in Director 11 you should use bitmaps with 16 or 32 bit color depth, as 8 bit ones may fail to render on Windows Vista.

Example:

-- Make a 64x64 icon out of member "monkey"

memNum = the number of member "monkey"

libNum = the castlibnum of member "monkey"

sprite(1).addIconXTV(libNum,memNum,45,64,64)

sprite(spriteReference).getIconXTV(iconID) - where spriteReference is the sprite channel containing the TreeView sprite and iconID is the positive integer ID of the icon in the tree sprite's icon list. Returns a string containing icon information for the specified icon ID or "" if the icon does not exist. Retuns information for the stored icon in the format:

castlibNumber memberNumber memberPalette widthInPixels heightInPixels

Example:

on checkCastlib spriteNum,lookForLibNum

-- Returns TRUE if all icons are from

-- castlibnum passed in

--

num = sprite(spriteNum).getNumIconsXTV()

```
num = num - 1

repeat with x = 1 to num

iconInfo = sprite(spriteNum).getIconXTV(x)

clib = value(word 1 of iconinfo)

if clib <> lookForLibNum then

return FALSE

end if

end repeat

return TRUE

end
```

sprite(spriteReference).setIconXTV(iconID,castlibNum,memberNum,memberPalette,pixelWidth,pixelHeight) - where spriteReference is the sprite channel containing the TreeView sprite, iconID is the positive integer ID of the icon in the tree sprite's icon list, castlibNum is the integer cast library number of the castlib containing the member to use for the icon, memberNum is the integer member number of the bitmap member to use for the icon, memberPalette is the member number in the cast library of the graphic's palette, pixelWidth is the integer width in pixels for the icon and pixelHeight is the integer height in pixels for the icon. Returns 0 if successful or a negative number if not. **Notice**: in Director 11 you should use bitmaps with 16 or 32 bit color depth, as 8 bit ones may fail to render on Windows Vista.

Resets the properties of an existing icon. If any of the tree's visible nodes use the icon, the visual properties of the nodes change immediately.

Example:

```
on changePalette spriteNum,pixWidth,pixHeight

-- Maps all existing icons to a new width and height

num = sprite(spriteNum).getNumIconsXTV()

num = num - 1

repeat with iconID = 1 to num

iconInfo = sprite(spriteNum).getIconXTV(iconID)
```

libNum = value(word 1 of iconInfo)

memNum = value(word 2 of iconInfo)

palNum = value(word 3 of iconInfo)

sprite(spriteNum).setIconXTV(iconID,libNum, memNum, palNum, pixWidth, pixHeight)

end repeat

end

sprite(spriteReference).getMaxIconSizeXTV(iconType) - where spriteReference is the sprite channel containing the TreeView sprite and iconType is an integer specifying which icon type to examine for the node. Must be one of the following values:

0: normal icon

1: highlighted icon

2: plus expander

3: minus expander

Returns a string containing the width and height or "ERRORCODE=-50" if an invalid iconType is specified. Returns the current settings for the specified icon type.

Example:

put sprite(spriteNum).getMaxIconSizeXTV()

-- "32 32"

sprite(spriteReference).setMaxIconSizeXTV(iconType,pixelWidth,pixelHeight) - where:

spriteReference is the sprite channel containing the TreeView sprite,

iconType is an integer specifying which icon type to examine for the node. Must be one of the following values:

0: normal icon

1: highlighted icon

2: plus expander

3: minus expander

pixelWidth is the integer width in pixels for the icon,

pixelHeight is the integer height in pixels for the icon.

Always returns 0.

Limits the height and width of the specified icon type. Icons linked to images larger than the specified height and width are scaled to fit. Icon images smaller than the height and width are not changed. The values set in this method override any set for individual icons with addIconXTV or setIconXTV if the limits of the icon are larger than those set in setMaxIconSizeXTV. This method is a more convenient way to scale a variety of different sized icon graphics. Use redrawXTV to refresh icons on stage after using this method.

This method does not scale scrollbar icons because they operate like "slipcovers", covering over the normal system scrollbar parts, and must be scaled to the size of the system scrollbar.

Example:

```
on scaleIcons treeSprite

-- plain folder max 32x32

sprite(treeSprite).setMaxIconSizeXTV(0,32,32)

-- highlighted folder max 32x32

sprite(treeSprite).setMaxIconSizeXTV(1,32,32)

-- plus expander max 16x16

sprite(treeSprite).setMaxIconSizeXTV(2,16,16)

-- minus expander max 16x16

sprite(treeSprite).setMaxIconSizeXTV(3,16,16)

-- update the stage

sprite(treeSprite).redrawXTV()

end
```

sprite(spriteReference).setGlobalIconIDXTV(iconType, iconID) - where:

spriteReference is the sprite channel containing the TreeView sprite,

iconType is an integer specifying which icon type to examine for the node. Must be one of the following values:

2: plus expander

3: minus expander

iconID is the positive integer ID of the icon in the tree sprite's icon list.

Returns 0 if successful or a negative number if there is an error.

Sets the icon graphic used throughout the tree for the expander icon type if the property has not yet been set individually for the node. You can also set this property for individual nodes with setIconIDXTV which will override the global icon. Use redrawXTV to refresh icons on stage after using this method. This method will change the icon of existing nodes with no expander icons yet set individually for them, and will assign the icon to newly created nodes after it is issued.

Example:

-- resets the tree's default plus expander icon to icon id 44

sprite(1).setGlobalIconIDXTV(2,44)

sprite(spriteReference).getGlobalIconIDXTV(iconType) - where:

spriteReference is the sprite channel containing the TreeView sprite,

iconType is an integer specifying which icon type to examine for the node. Must be one of the following values:

2: plus expander

3: minus expander

Returns the integer icon id for the specified icon type. Returns the global icon ID set for the specified icon type. If a global icon ID has not been set for the type, returns -1.

Example:

expanderPlus = sprite(1).getGlobalIconIDXTV(2) sprite(spriteReference).setAllNodesIconIDXTV(iconType,iconID) - where:

spriteReference is the sprite channel containing the TreeView sprite,

iconType is an integer specifying which icon type to examine for the node. Must be one of the following values:

0: normal icon

1: highlighted icon

2: plus expander

3: minus expander

iconID is the positive integer ID of the icon in the tree sprite's icon list.

Returns 0 or a negative number if there has been an error.

Resets the icon for the specified icon type for existing nodes. The difference between setGlobalIconIDXTV and setAllNodesIconIDXTV is that setGlobalIconIDXTV changes the default icon used when new nodes are created while setAllNodesIconIDXTV only alters the icons of existing nodes.

Example:

-- sets the normal icon of existing nodes to icon 88

sprite(1).setAllNodesIconIDXTV(0,88)

sprite(spriteReference).setClickboxColorsXTV(frameRed,frameGreen,frameBlue,fillRed,fillGreen,fillBlue) - where spriteReference is the sprite channel containing the TreeView sprite, frameRed is the integer RGB red value, frameGreen is the integer RGB green value, frameBlue is the integer RGB blue value, fillRed is the integer RGB red value, fillGreen is the integer RGB green value, fillBlue is the integer RGB blue value. Always returns 0. Sets the colors used by the default expander box icon. Does not change custom expander icons.

Example:

-- set expander box outline to blue and

-- interior to green

sprite(1).setClickboxColorsXTV(0,0,65535,0,65535,0)

CONNECTOR LINE AND SPACING PROPERTIES

The following methods control the layout options of the tree as a whole. Keep in mind that individual node icons of varying sizes will also impact layout. For a more uniform layout, use setMaxIconSizeXTV to scale down larger icon graphics to one size.

sprite(spriteReference).setLineModeXTV(lineType) - where spriteReference is the sprite channel containing the TreeView sprite and lineType is the integer type number, and one of the following:

0: solid

1: dotted

2: none

Always returns 0.

Controls the type of line used for the connecting lines between nodes. Tree updates immediately when this property is set. Use setMainColorsXTV to set the connecting line color.

Example:

-- remove connecting lines

sprite(1).setLineModeXTV(2)

sprite(spriteReference).getLineModeXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. Returns the integer value of the current line mode set for the tree. Returns the current line type used for the connecting lines between nodes.

Example:

sprite(1).currentLineType = getLineModeXTV()

sprite(spriteReference).setMainColorsXTV(lineRed,lineGreen,lineBlue) - where spriteReference is the sprite channel containing the TreeView sprite, lineRed is the integer RGB red value, lineGreen is the integer RGB green value and lineBlue is the integer RGB blue value. Always returns 0. Sets the color of the connecting lines between nodes. Setting this property changes the color of connecting lines already drawn on stage immediately.

Example:

-- sets connecting line color to blue

sprite(1).setMainColorsXTV(0,0,44000)

sprite(spriteReference).getContentSizeXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. Returns a string containing current width and height of a tree's content area, separated by a space.

Returns a string containing the width and height in pixels of a tree's content area. Typical return looks like this: "220 447". The content area is the area occupied by a tree's icons, not the size of the TreeView sprite. If the user expands a tree node, the height of the content area will change but the height of the TreeView sprite will not. If the content area extends beyond the bounds of the TreeView sprite, and you have scrollbars turned on, the scrollbars will activate, otherwise the user will not be able to access any nodes outside of the boundaries of the TreeView sprite.

If you change the content size of the tree inside a callback handler by using commands like setExpandedXTV or by changing the icon of a node to an icon of a different size, the value returned by getContentSize will not reflect the new size of the tree if you call it inside the same callback handler. You must call it after the callback handler has finished to get the new tree size.

Example:

on treeViewSingleClick refcon, nodeID, whichPart,spriteChan

-- Makes scrollbars visible only when content

-- exceeds sprite boundaries

--

-- Assumes that tree sprite's refcon property has

-- been set to its sprite number

```
--

sprite(refcon).contentSize = getContentSizeXTV( )

contentWidth = value(word 1 of contentSize)

contentHeight = value(word 2 of contentSize)

put refcon

if (contentWidth > the width of sprite refcon) or (contentHeight > the height of sprite refcon) then

sprite(refcon).setScrollbarUsageXTV(TRUE)

else

sprite(refcon).setScrollbarUsageXTV(FALSE)

end if

end
```

sprite(spriteReference).setContentOffsetXTV(horizontalPixels,verticalPixels) - where spriteReference is the sprite channel containing the TreeView sprite, horizontalPixels is the number of pixels to offset the content toward left, verticalPixels is the number of pixels to offset the content upward. Always returns 0.

Moves content area within TreeView sprite. Positive numbers move the content up and to the left (outward). Negative numbers move the content down and to the right (inward).

After setting this property with Lingo, the scrollbars consider point 0,0 (top/left) of the tree to be the beginning of the offset. Always call resetScrollbarsXTV after adjusting this property for a sprite with scrollbars on. Otherwise, if the content shifts so that the scrollbars become inactive, the user may be unable to access the content.

```
----------- top of tree, point 0,0

| > TOPNODE

|   > CHILD
```

After setContentOffsetXTV(sprite treesprite, 0, -20)

```
----------- top of tree, point 0,0
```

```
|

|

| > TOPNODE

|   > CHILD
```

Note: This command was intended to allow developers to add white space around the nodes at the top of the tree. It can be used to scroll the tree down to show a particular node, but the scrollbars will consider point 0,0 to be the start of the offset so the user will not be able to scroll above the current offset.

Example:

```
-- Moves the tree content area 10 pixels in from

-- left edge of TreeView sprite edge and 10 pixels

-- down from top of TreeView sprite edge

sprite(1).setContentOffsetXTV(-10,-10)
```

sprite(spriteReference).setSpacingXTV(indent,iconSideMargin,iconTopMargin,expanderSideMargin,expanderTopMargin)
- where spriteReference is the sprite channel containing the TreeView sprite, indent is the integer pixels to indent each level of the tree, iconSideMargin is the integer pixels to indent the icons to the right of the vertical line, iconTopMargin is the integer pixels of the vertical space between icons, expanderSideMargin is the integer pixel width of the expander icon, expanderTopMargin is the integer pixel height of the expander icon. Always returns 0.

Adjusts the spacing between tree icons. The width and height of the expander icon can only be adjusted for the TreeView default icon. Setting values for either property will not change the size of a custom expander icon. Negative values are allowed. Default values are:

indent: 16

iconSideMargin: 4

iconTopMargin: 4

expanderSideMargin: 9

expanderTopMargin: 9

Example:

-- Keep the defaults for other props but change the

-- vertical space between icons

sprite(1).setSpacingXTV(16,4,10,9,9)

sprite(spriteReference).getSpacingXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. Returns a string containing the tree's current spacing settings separated by spaces. Returned string has the following format:

indent iconSideMargin iconTopMargin expanderSideMargin expanderTopMargin

Example:

put sprite(1).getSpacingXTV()

-- "16 4 4 9 9"

SCROLLBAR PROPERTIES

The following methods control the display and appearance of the scrollbars.

sprite(spriteReference).setScrollbarUsageXTV(onOrOff) - where spriteReference is the sprite channel containing the TreeView sprite and onOrOff is 0 for off, 1 for on. Always returns 0. Shows or hides the scrollbars for a TreeView sprite. If the tree's content area is not large enough to scroll, TreeView will automatically draw the scrollbars in an inactive state. Using this method resets the content offset to 0,0.

Example:

-- Hide scrollbars

sprite(1).setScrollbarUsageXTV(0)

sprite(spriteReference).setScrollbarDataXTV(castlibNum,startingMemberNum,0) - where spriteReference is the sprite channel containing the TreeView sprite, castlibNum is the integer castlib number containing the scrollbar icons, startingMemberNum is the integer member number of the first scrollbar icon and notUsed is reserved for future use, so always pass 0. Returns 0 if successful or a negative number if not.

Sets the images to use for the various parts of the vertical and horizontal scrollbar. Contrary to node icons, the starting image for the scrollbar is specified directly as a cast member. To prepare to use this method you must first place the 17 graphics that make up the different parts of the scrollbar in contiguous cast slots in the following order. They do not have to start at cast member 1, but they do have to be in the order specified, with no unrelated cast members in between them.

Member order in cast for scrollbar graphics

Position	Scrollbar part
1	Horizontal bar, left arrow, active/up
2	Horizontal bar, left arrow, down
3	Horizontal bar, left arrow, inactive
4	Horizontal bar, right arrow, active/up
5	Horizontal bar, right arrow, down
6	Horizontal bar, right arrow, inactive
7	Vertical bar, top arrow, active/up
8	Vertical bar, top arrow, down
9	Vertical bar, top arrow, inactive
10	Vertical bar, bottom arrow, active/up
11	Vertical bar, bottom arrow, down
12	Vertical bar, bottom arrow, inactive
13	Horizontal thumb up
14	Horizontal thumb down
15	Vertical thumb up
16	Vertical thumb down

17 Corner box

The horizontal scrollbar height and the vertical scrollbar width will adjust to accomodate the size of the thumb and arrow icon images, however, it is better to make the images as close to the size of the usual system scrollbar parts as possible. Otherwise you could create scrollbars that don't align properly. The corner box image will be cropped to the size of corner space created by the two scrollbars.

Example:

-- Set the scrollbar images to member 83 and following

sprite(1).setScrollbarDataXTV(1, 83, 0)

sprite (spriteReference).setScrollbarColorsXTV (inactiveRed,inactiveGreen,inactiveBlue,frameRed,frameGreen,frameBlue,activeRed,activeGreen,activeBlue) - where spriteReference is the sprite channel containing the TreeView sprite, inactiveRed is the integer RGB red value, inactiveGreen is the integer RGB green value, inactiveBlue is the integer RGB blue value, frameRed is the integer RGB red value, frameGreen is the integer RGB green value, frameBlue is the integer RGB blue value, activeRed is the integer RGB red value, activeGreen is the integer RGB green value, and activeBlue is the integer RGB blue value. Always returns 0.

Sets the color used to draw the frame outline of the scrollbars and the color that fills the interior of the scroll bar tracks. Note: this command has changed since version 1.0. It requires 3 more parameters to specify the color of the active scrollbars.

Example:

-- Configure scrollbars for black outline, blue active, gray inactive

sprite(1).setScrollbarColorsXTV (20000, 20000, 20000, 0, 0, 0,0, 0, 65535)

sprite(spriteReference).resetScrollbarsXTV(0) - where spriteReference is the sprite channel containing the TreeView sprite and reservedForFutureUse is reserved for future use, so always pass 0. Always returns 0. Repositions the tree content and scrollbars so tree content is at top/left. Scrolls vertical scrollbar all the way up and horizontal scrollbar all the way left.

Example:

sprite(1).setExpanded(FALSE)

sprite(1).resetScrollbarsXTV(0)

TEXT PROPERTIES

You can replace the default TreeView system font used to display node text with any font resident on the user's system or with a Director 7 embedded font. You set the font for each node's text individually using setFontIDXTV. There is no tree-wide method for setting the font, however you can set the font property at the node's creation when you set other properties for the node such as its text and icon.

TreeView icon methods do not reference fonts directly by name. They look to an internal TreeView font list to get font information. You use the addFontXTV method to add a system or Director 7 font to the font list to make it available for use by any of the other font methods. As each font is added to the font list TreeView assigns it a unique font ID, similar to the node ID's of TreeView nodes. You must use font ID's to refer to fonts in the font list, not font names.

The font list stores a string with each font ID that specifies its properties in the following format:

fontName pointSize styleNumber

fontName: name of font associated with this font id

pointSize: point size to use for this font id

styleNumber: one of the following specifies font style for this font id

0: plain

1: bold

2: italic

3: bold and italic

This is what a typical return from the getFontXTV method looks like. Font 1 in the font list is 18 point Chicago bold.

put sprite(1).getIconXTV(1)

-- "Chicago 18 1"

Font ID 0 is always the TreeView default system font. It cannot be modified or deleted. A TreeView sprite will always return at least 1 from getNumFontsXTV even if no custom fonts have been added, because font 0 is present.

sprite(spriteReference).addFontXTV(fontName,fontSize,fontStyle) - where spriteReference is the sprite channel containing the TreeView sprite, fontName is the string containing the system name for the font, fontSize is the integer font point size and fontStyle is the integer font style, one of the following:

0: plain

1: bold

2: italic

3: bold and italic

Returns the font ID in the font list of the newly-added font.

Adds a font to TreeView's internal font list. Since fonts must be added with a particular point size and style, to use both Courier bold and Courier plain, for example, make two entries in the font list. You cannot add a font with all of the same parameters twice. In other words you can't add "Arial 12 plain" twice. If a font with the same specs is already in the list, that font's ID is returned and no action is taken.

Example:

```
--Adds 18-point Helvetica bold to the font list

sprite(1).addFontXTV("Helvetica",18,1)
```

sprite(spriteReference).setFontXTV(fontID,fontName,fontSize,fontStyle) - where spriteReference is the sprite channel containing the TreeView sprite, fontID is the integer font ID of the existing font in the font list, fontName is the string containing the system name for the font, fontSize is the integer font point size and fontStyle is the integer font style, one of the following:

0: plain

1: bold

2: italic

3: bold and italic

Returns 0 or a negative number if unsuccessful.

Modifies the properties of a font in the font list. If the font is being used by any node, the node's text changes to use the new font. You cannot change the properties of font 0, the default font. If you add at least one font to the font list and create all nodes initially to use that font instead of the default font, you can change the font for all tree nodes easily by using setFontXTV to change the properties of the existing font.

Example:

-- Change the properties of font 2 and hence the

-- font of all nodes using font ID 2

sprite(1).setFontXTV(2,"Charcoal",12,0)

sprite(spriteReference).getFontXTV(fontID) - where spriteReference is the sprite channel containing the TreeView sprite and fontID is the integer font ID of the existing font in the font list. Returns a string containing font properties set for a font if it exists in the font list or "" if it doesn't. The return string has the following format:

"fontName pointSize style"

Example:

put sprite(1).getFontXTV(1)

-- "Chicago 18 0"

sprite(spriteReference).getNumFontsXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. Returns the number of fonts currently in TreeView's font list. Since the font list always contains font 0, the default font, this number can never be less than one. Since the font ID's start at 0, subtract 1 from the return value to get the ID of the last font in the list.

Example:

on displayFontList treeSprite

```
lastFontID = sprite(treeSprite).getNumFontsXTV() - 1

repeat with x = 1 to lastFontID

put x & ") " & sprite(treeSprite).getFontXTV(x)

end repeat

end



displayFontList(2)

-- "1) Apple Chancery 10 3"

-- "2) Arial 12 1"

-- "3) Capitals 14 2"
```

sprite(spriteReference).setTextHiliteColorXTV(textRed,textGreen,textBlue) - where spriteReference is the sprite channel containing the TreeView sprite, textRed is the integer RGB red value, textGreen is the integer RGB green value, and textBlue is the integer RGB blue value. Always returns 0. Sets the highlight color that displays over node text when the user clicks on a node. TreeView automatically highlights a node's name when the user clicks on the node. The default highlight color is blue.

Example:

```
-- Set the text highlight color to red

sprite(2).setTextHiliteColorXTV(65535,0,0)
```

MISCELLANEOUS PROPERTIES

The following properties determine how the TreeView sprite is identified in the callback handlers and update its display or stage.

sprite(spriteReference).getVersionXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. Returns the string containing the version number of TreeView Xtra currently loaded.

Example:

```
put sprite(2).getVersionXTV()

-- "3.0"
```

sprite(spriteReference).setRefconXTV(newID) - where spriteReference is the sprite channel containing the TreeView sprite and newID is the integer to use to identify this tree. Always returns 0. Sets the id number that is passed to TreeView callback handlers to identify the tree that the user clicked. Default is 0. This property can be useful to distinguish between different trees that appear in the same sprite channel at different times or in different frames.

Example:

```
on beginSprite me

-- Sprite behavior that sets this

-- tree's refcon to its sprite number

treeSpriteNum = the spritenum of me

sprite(treeSpriteNum).setRefconXTV( treeSpriteNum)

end
```

sprite(spriteReference).getRefconXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. Returns the current refcon number of the TreeView sprite. Refcon will be 0 if it has never been set.

Example:

```
curRefCon = sprite(1).getRefconXTV()
```

sprite(spriteReference).redrawXTV(spriteReference) - where spriteReference is the sprite channel containing the TreeView sprite. No return. Updates the TreeView sprite display after visual tree properties have been changed.

Example:

```
sprite(1).redrawXTV()
```

TREEVIEW XTRA HELP: CALLBACK HANDLERS

When a user action occurs over one of TreeView's nodes you can choose to have a message sent to a callback handler. The callback handler for each event must use the designated name for that event's handler and can reside in a sprite, frame, cast member or movie type of script. When a TreeView event occurs, such as "single click" it is sent to the script levels in the following order, until it hits a level containing a callback handler for it.

sprite --> cast member --> frame --> movie

The designated callback handlers are as follows:

```
on treeViewSingleClick refcon, nodeId, whichPart, spriteChannel

end
```

Called when the user clicks within the bounding box of a node. Detects event on icon or name.

```
on treeViewDoubleClick refcon, nodeId, whichPart, spriteChannel

end
```

Called when the user double-clicks within the bounding box of a node. Detects event on icon or name.

```
on treeViewMouseEnter refcon, nodeId, whichPart, spriteChannel

end
```

Called once when the mouse enters the bounding box of the node, if the mouse is up. Detects events on expander, icon, name and whitespace.

```
on treeViewMouseExit refcon, nodeId, whichPart, spriteChannel

end
```

Called once when the mouse leaves the bounding box of the node, if the mouse is up. The whichpart parameter is not used and always reports whitespace.

```
on treeViewMouseMove refcon, nodeId, whichPart, spriteChannel

end
```

Called while the mouse is within the bounding box of a node. Detects events on expander, icon, name and whitespace.

```
on treeViewMouseDown refcon, nodeId, whichPart, spriteChannel

end
```

Called when the user mouses down within the bounding box of a node. Detects events on expander, icon, name and whitespace.

Each callback handler receives the same information about the user event:

refcon: custom number that is not a sprite number but is used to distinguish between TreeView sprites

id: node ID of node user action happened on whichPart: 0: expander area of node 1: icon area of node, 2: text area of node 3: whitespace within node bounding box

The SingleClick and DoubleClick events only report icon or text for whichpart.

channel: sprite number of the treeview sprite the action happened on

To customize which handlers should be activated, use setAllNodesMouseStatusRequestXTV to set event-handling tree-wide, or setMouseStatusRequestXTV to set it per individual node. Both SingleClick and DoubleClick are initially on by default. Note that the mouseEnterFlag argument of those methods turns on both the MouseEnter and MouseExit callback handlers.If both SingleClick and DoubleClick are enabled, the first click of a user's double-click action triggers the SingleClick callback.

All of the programming of TreeView sprite interactivity happens inside of a callback handler. You can use the information passed in to the handler to read or change tree properties in response to user actions.

USING CALLBACK HANDLERS IN BEHAVIORS

When a TreeView callback handler is included in a behavior, it passes back a parameter in the first position that is a refcon reference, not a reference to the behavior.The parameter that is usually "me" instead contains a reference to the TreeView refcon that is receiving the event. That means that a TreeView callback handler inside a behavior will not know about the behavior's properties. They will all come up as Void. To work around this, call another method inside the behavior, and pass it any information it may need that came from the TreeView callback handler. Do something like this:

```
property myProp



on beginSprite me

myProp = 8



end
```

```
on treeViewSingleClick refcon, nodeID, whichpart,spriteChan

-- myProp will be Void

sendSprite(spriteChan,#anotherHandler,refcon,nodeID,whichpart)

end


on anotherHandler me,refcon,nodeID,whichpart

-- "me" the first param, refers to behavior again

-- so myProp will be recognized, and we still have the values returned

-- from the callback handler

--

if myProp = 8 then

alert(string(nodeID))

end if

end
```

IMPORTANT NOTE

You cannot do a "go frame" from inside a callback handler to a new frame that does not have the same TreeView sprite in the same channel or Director will crash. The same thing will happen with "go movie". One workaround is to keep the TreeView sprite in the same channel throughout the movie and move it offstage in the frames where it should not show. Another workaround is to set a flag in the callback handler and have an exitframe handler execute the "go" when the flag changes.

EXAMPLE

Here is a simple example of callback handling code:

```
on treeViewDoubleClick refcon, id, whichPart,channel

nodeID = id

treesprite = channel

-- Retrieve the path to an application stored in the node's

-- user storage area previously, with setUserStringXTV,

-- and launch it

--

programLaunchPath = sprite(treesprite).getUserStringXTV(nodeID,0)

open programLaunchPath

end
```

TREEVIEW XTRA HELP: EVENT HANDLING

TreeView handles mouseUp and mouseDown events on TreeView sprites completely independently from Director. That means that Director never receives these events for a TreeView sprite and mouseUp or mouseDown handlers in sprite or cast scripts placed on the TreeView sprite or cast member will not be called. You can trap normally for other events in behavior or cast scripts on TreeView sprites/members.

TREEVIEW XTRA HELP: ERROR CODE LIST

Return Value	Meaning
0	No error occurred
-1	Call could not complete successfully / node does not exist.
-2	Out of memory
-50	One or more of the parameters to the method was wrong
-38	Tried to add a child, and setMagicXTV was not called with the correct serial number. Returned from methods which add child nodes
Any other negative number	Mac system or Windows system error. This should be a rare occasion. Consult Mac or Win system documentation.

TREEVIEW XTRA HELP: COLOR

All TreeView methods that require color specifications use RGB values, and expect red, green and blue values in the range of 0 to 65535. If you are more comfortable with 0 - 255 values, the way Director 7 handles RGB, you can convert those values to TreeView values like so:

```
on rgbConvert colorList

-- Expects a list like this [r,g,b] where

-- rgb are the color values in scale 0 - 255

-- Returns a list in the same format with

-- color values in scale 0 - 65535

--

-- EX: put rgbConvert( [255,150,0] )

-- [65535, 38550, 0]

--

newList = []

repeat with x = 1 to 3

convertedColor = (65535/255) * getAt(colorList,x)

add newList,convertedColor

end repeat

return newList

end
```

TREEVIEW XTRA HELP: JAVASCRIPT

XTRAS AND JAVASCRIPT

Director MX 2004 added JavaScript as an alternative scripting language, and it is also available in Director 11. The syntax in the methods doc for TreeView Xtra works for both Lingo and JavaScript. For more info, see the tech note on JavaScript and Xtras.

TREEVIEW XTRA HELP: SHOCKWAVE

TreeView Xtra can be used in Shockwave: the distribution package provides packaged Xtras that are downloaded automatically to the user's machine, and installed on demand. Please consult Adobe's web site for a complete overview of the Xtras automated download mechanism: read the Shockwave Xtras downloading overview technote. The basic steps required to make TreeView Xtra available for download are outlined below.

To create a Shockwave movie that will auto-download the Xtra to the user's hard drive you must do the following, in this order:

1. Upload the packaged Xtra files to your web server

2. Modify the entry for TreeView Xtra in file xtrainfo.txt to point to the packaged Xtra files on your server

3. Do Modify -> Movie -> Xtras, select TreeView Xtra, and check the "Download if Needed" option

Once you have completed steps 1 and 2, you can create other Shockwave movies by doing only step 3.

PACKAGED FILES

Your TreeView Xtra archive contains a subfolder called "Shockwave". There are four files inside it:

TreeView.w32 - Win 32 package

TreeView.ppc - Mac Classic package

TreeView.carb - Mac Carbon package

TreeView.xpku - Mac Universal Binary package

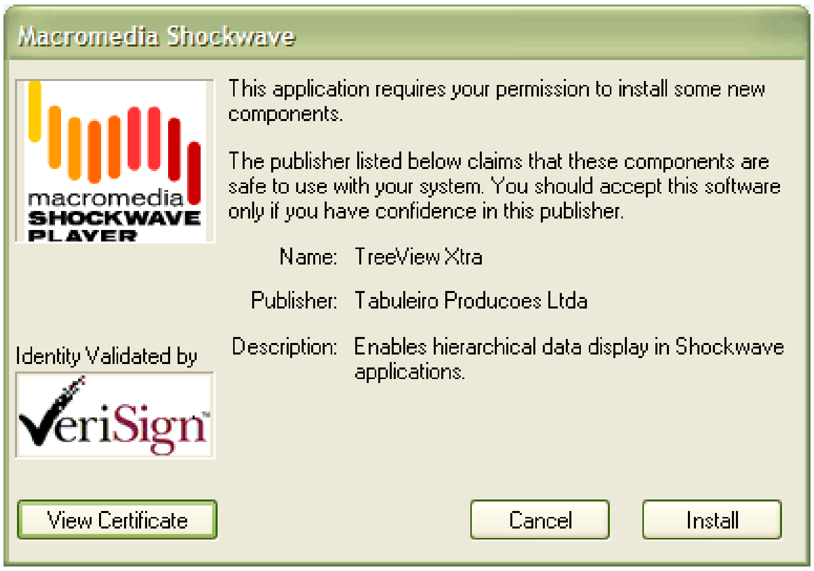
All packages contain the TreeView Xtra for that platform. Depending on the user's platform, a package autodownloaded to the user's hard drive will install the correct TreeView Xtra for the user's platform into their Shockwave support folder.

If, for some reason, you choose not to make your Shockwave movies autodownload the package files, you can have the user install the right Xtra for their platform into the Shockwave support folder manually.

Upload all package files to the same directory on your web server. Use a "binary" or "raw", not "text" transfer. If the packages are uploaded to two different directories, autodownloading will not work. Do not rename the package files.

If you are going to distribute TreeView Xtra with Shockwave movies, we recommend that you use your own web server to do so. Packages are available at Tabuleiro's download services, but we reserve the right to refuse access, without notice, to any referring URL that generates excessive traffic.

The TreeView Xtra packages included with the download have been signed and packaged by Tabuleiro, and will present the following security message to users of your Shockwave movies when they are installed for the first time:



Macromedia Shockwave

This application requires your permission to install some new components.

The publisher listed below claims that these components are safe to use with your system. You should accept this software only if you have confidence in this publisher.

Name:	TreeView Xtra
Publisher:	Tabuleiro Producoes Ltda
Description:	Enables hierarchical data display in Shockwave applications.

Identity Validated by

You may choose to repackage TreeView Xtra and sign it with your own Verisign certificate. You might want to do this if you want your own company name to appear in the auto-download dialog box the user sees when an auto-download is initiated. Adobe is the best source of information on applying for a Verisign certificate and packaging files.

XTRAINFO.TXT

The text file xtrainfo.txt resides in your Director authoring directory. It contains information about Xtras such as file version names for an Xtra on both platforms and the URL for the packages. The information contained in xtrainfo.txt is saved with each movie you create and used by projectors and Shockwave.

You must create an entry for TreeView Xtra in your xtrainfo.txt file that specifies the URL on your server for the package files. The last part of the path will always be "TreeView". That specifies the filename of the packages within the directory, without the file extension. Do not include a file extension at the end of the path.

```
[#namePPC:"TreeView", #nameW32:"TreeView.x32",
#package:"http://www.domain.com/folder/TreeView"]
```

Make sure that the line above does not contain any return character after you paste it into your xtrainfo.txt file. Open your text editor wide and make sure the line does not wrap. If the opening and closing brackets are not on the same line, Director will not be able to create a valid list from the entry and the "Download if needed" button will be dimmed for TreeView Xtra in Director.

If you edit xtrainfo.txt while Director is open you should quit and restart Director to read in the changed information in xtrainfo.

EDITING THE MOVIE'S XTRAS LIST

Open the Director movie that you want to save as Shockwave. Choose Modify -> Movie -> Xtras and add TreeView. Select TreeView from the list and check the "Download if needed" option. Director will initiate an internet connection and look for the packages at the URL you specified in xtrainfo.txt.

If Director finds the packages, it will transfer information about the package contents for both platforms such as file names and version numbers and embed the information into your Director movie. An informational dialog box will appear that tells you that the packages for both platforms are "downloading". The packages themselves are not downloading, just information about them that the Shockwave movie will need later to compare the version of the Xtra the user possibly already has to the version currently on the server in order to determine if autodownloading is necessary. The Director movie needs information about both platforms because it may find itself running on either platform once it is on the web.

Once "downloading" of the packages has finished, save the movie, then publish as Shockwave. The finished Shockwave movie can reside at any URL. It does not have to be in the same directory or even on the same server as the packaged Xtras.

If a connection cannot be opened, or the packages cannot be found at the specified location, Director will uncheck the "Download as needed" option automatically. You must have a successful connection for the box to remain checked. A Shockwave made out of a Director movie with the "Download as needed" button unchecked will not autodownload TreeView Xtra.

TREEVIEW XTRA HELP: LIMITATIONS

Icon bitmaps: When running in Director 11 it is recommended to use 16 or 32 bit depth bitmaps for all icon operations in TreeView. Some 8 bit bitmaps might fail to display correctly on Windows Vista.

Unicode strings: When running in Director 11 TreeView will attempt to convert and render the supplied UTF8 string data. However, not all glyphs are available and some double byte languages might fail to load.

International fonts: TreeView does not support double-byte fonts. One user has reported mapping errors with some single-byte Hebrew fonts.

Callbacks: If both SingleClick and DoubleClick are enabled, the first click of a user's double-click action triggers the SingleClick callback.

Moving treeview sprite offstage: If you move a treeview sprite outside the stage boundaries, its old rect will still receive treeview events unless you also set its sprite's visibility to FALSE

```
sprite(treesprite).loc = point(-1000,-1000)

sprite(treesprite).visible = FALSE
```

Altering the size of the tree: If you alter the size of the tree content using Lingo, for instance with setExpandedXTV, or by changing the size of node icons, you must then use resetScrollbarsXTV to re-calibrate the scrollbars.

Scrolling to a particular node: There is no built-in command to scroll the tree down to a particular node. Although it is possible to do this using getNodeLocationXTV and setContentOffsetXTV, the scrollbars were not designed for this and will continue to consider the top of the tree as the beginning of the offset. If you use Lingo to scroll the tree to a particular node, you must also create your own scrollbars using sprites.

Scaling: TreeView does not completely support scaling. When shown on a magnified stage, the bounding box of a TreeView sprite scales, but the contents do not.

Callback crash: There are some Lingo operations that cannot be performed inside TreeView callback handlers because they cause a crash. This is a limitation of sprite Xtras in general, not TreeView Xtra in particular.

Number of nodes: TreeView can create and manage up to a million nodes per tree. That is the internal logical limit. However, available memory, CPU speed and graphic card performance will determine how large a tree your application can actually build and manage with acceptable performance. TreeView was tested at up to 10,000 nodes per tree.

The memory requirement per unnamed node is between 200 and 600 bytes depending on the total number of nodes in the tree. That does not include any additional storage taken up by the node name or custom user strings set for the node with setUserStringXTV.

Tree properties: All of the properties of a TreeView tree are stored with its sprite. If you move to a frame that does not contain the TreeView sprite, the properties will be reset when you return to the frame. If you want to maintain TreeView sprite properties between frames where the sprite should not appear, move it offstage (locV = -1000) but keep it in the sprite channel.

TREEVIEW XTRA HELP: HOW TO ORDER & REGISTER

The unregistered version of TreeView Xtra is fully-functional and may be used for evaluation, nonprofit and educational purposes only: commercial distribution is strictly prohibited. A registered version of the Xtra can be used in commercial products, and may be purchased online at xtras.tabuleiro.com, using a secure server. At our web site you can also consult our purchase policy, purchase instructions, payment, delivery and security methods.

If you decide to buy the Xtra you don't need to download a new copy of the software. After your order is processed you will receive an e-mail with a serial number to register the software you've already installed on your machine.

To register the Xtra you should use the setMagicXTV() sprite function, before any other TreeView Xtra function is used on a TreeView sprite . More information about specific syntax can be found at the Methods Documentation page. Please keep your serial number archived for future reference.

TREEVIEW XTRA HELP: LICENSING & AVAILABILITY

TreeView Xtra is a commercial product. Current price and updated information can be found at xtras.tabuleiro.com. If your product provides printed documentation and package we ask you to kindly include the following copyright information:

No royalty-fees are required for a distribution of the Xtra with your product.

TREEVIEW XTRA HELP:
TECHNICAL SUPPORT

Please use the Your Account section available at our web site xtras.tabuleiro.com to submit your questions. The site also contains Technotes and other resources that can help you identify and solve the most common problems quickly.