MUI Royale 1.7

Easy GUI Creation For You Kings and Queens of MUI

Andreas Falkenhahn

Table of Contents

1 General information

1.1 Introduction

MUI Royale is a plugin for Hollywood that allows you to easily create MUI GUIs with Hollywood. The GUI layout can be conveniently defined using an XML file that is converted into a MUI GUI by MUI Royale on the fly. It just doesn't get any easier!

MUI Royale supports over 40 MUI classes including popular third-party classes like TextEditor.mcc and TheBar.mcc. Creating and managing menustrips is also fully supported. The highlight of MUI Royale, however, is certainly its inbuilt Hollywood MUI class. This class allows dynamic embedding of complete Hollywood displays into MUI GUIs which can be used to combine the best of Hollywood and MUI into one powerful application.

MUI Royale comes with extensive documentation in various formats like PDF, HTML, AmigaGuide, and CHM that describes MUI programming basics in detail and provides a convenient function and class reference. A step-by-step tutorial that guides you to your first MUI program is also included. On top of that, almost 20 example scripts are included in the distribution archive, many of which are direct ports from the MUI 3.8 SDK by Stefan Stuntz, but there are also original developments that show how to create video and song players using the combined power of Hollywood and MUI.

All this makes MUI Royale the truly royal MUI experience, carefully crafted for you kings and queens of MUI!

1.2 Terms and conditions

MUI Royale is © Copyright 2012-2017 by Andreas Falkenhahn (in the following referred to as "the author"). All rights reserved.

The program is provided "as-is" and the author can not be made responsible of any possible harm done by it. You are using this program absolutely at your own risk. No warranties are implied or given by the author.

This plugin may be freely distributed as long as the following three conditions are met:

1. No modifications must be made to the plugin.
2. It is not allowed to sell this plugin.
3. If you want to put this plugin on a coverdisc, you need to ask for permission first.

This software uses the Magic User Interface (MUI) which is (C) Copyright 1992-97 by Stefan Stuntz. See Section A.1 [MUI license], page 299, for details.

This software uses Expat (C) Copyright 1998, 1999, 2000 Thai Open Source Software Center Ltd and Clark Cooper. (C) Copyright 2001, 2002, 2003, 2004, 2005, 2006 Expat maintainers. See Section A.2 [Expat license], page 299, for details.

This program uses TextEditor.mcc by Allan Odgaard and the TextEditor.mcc Open Source Team. See Section A.3 [LGPL license], page 299, for details.

This program uses TheBar.mcc by Alfonso Ranieri and the TheBar.mcc Open Source Team. See Section A.3 [LGPL license], page 299, for details.

This program uses codesets.library by Alfonso Ranieri and the codesets.library Open Source Team. See Section A.3 [LGPL license], page 299, for details.

1.3 Requirements

– Hollywood 5.2 or better
– MUI 3.8 or better
– 68020+ or PowerPC
– codesets.library for UTF-8 support
– Toolbar class requires TheBar.mcc 26.11 or better
– Texteditor class requires TextEditor.mcc 15.47 or better
– optional: CyberGraphX or Picasso96 for RTG screens
– optional: guigfx.library and render.library for palette screens

1.4 Support for palette screens

Starting with MUI Royale 1.4, the plugin can also be used on palette-based screens, i.e. on screens in depths of 8-bit and below. This feature, however, requires guigfx.library and render.library to be installed. Otherwise, MUI Royale will fail to work on palette-based screens.

If your application doesn't only use MUI windows, but also needs to open normal Hollywood displays in addition to the MUI windows, you also have to install the Plananarama plugin which allows Hollywood to run on palette-based screens and you need to call `@REQUIRE` on

Plananarama before you call `@REQUIRE` on MUI Royale, i.e. your script has to start like this:

```
@REQUIRE "plananarama"
@REQUIRE "muiroyale"
```

If your script only uses MUI windows, however, you don't need Plananarama. guigfx.library and render.library are sufficient then. If you wish to open Hollywood displays on palette-based screens as well, though, make sure to install Plananarama first.

2 About MUI Royale

2.1 History

Please see the file `history.txt` for a complete change log of MUI Royale.

2.2 Future

Here are some things that are on my to do list:

- support for drag'n'drop
- support for more MUI classes

2.3 Frequently asked questions

This section covers some frequently asked questions. Please read them first before asking on the mailing list or forum because your problem might have been covered here.

Q: My GUI is not resizable. Isn't one of the big advantages of MUI that its GUIs are always resizable and MUI takes care of this automatically?

A: Yes, but there are some things that you need to keep in mind for the resize feature to work correctly. If there is a gadget in your GUI that has a fixed size, you need to pad it using resizable `<rectangle>` objects on its sides. Then your GUI will be resizable again. For example, imagine you have a 64x64 `<image>` object in a horizontal group in your window. MUI won't be able to resize this window unless you add rectangle objects to the sides of your image object because MUI needs to find an object that can be resized so you need to take care that non-resizable objects in your GUI are always padded with resizable ones. By the way, be careful with the `<label>` tag: Objects of label class are actually not resizable! If you want resizable labels, just use `<text>` instead.

Q: Why are labels not resizable then?

A: Honestly, I don't know. There is a `Label.FreeVert` attribute but the corresponding `Label.FreeHoriz` is missing. I do not really know why Stefan Stuntz did it this way but I am sure there is some rationale behind it.

Q: Why are there no Application.Author, Application.Copyright, Application.Description, Application.Title, and Application.Version attributes?

A: These attributes have been deliberately left out to avoid redundancies with Hollywood. Hollywood already offers the `@APPAUTHOR`, `@APPCOPYRIGHT`, `@APPDESCRIPTION`, `@APPTITLE`, and `@APPVERSION` preprocessor commands. The contents of these preprocessor commands will be automatically passed to the MUI application object when it is created.

2.4 Credits

MUI Royale was written by Andreas Falkenhahn. Thanks have to go to Stefan Stuntz for his wonderful MUI toolkit, Alfonso Ranieri for TheBar.mcc and codesets.library, Allan

Odgaard for TextEditor.mcc, and the TheBar.mcc and TextEditor.mcc Open Source Teams for maintaining these classes and fixing old bugs.

If you need to contact me, please send an email to andreas@airsoftsoftwair.de or use the contact form at http://www.hollywood-mal.com.

3 Fundamental MUI concepts

3.1 Application tree

A MUI application consists of a (sometimes very) big object tree. The root of this tree is always an instance of application class, called application object. This application object handles the various communication channels such as user input through windows, ARexx commands or commodities messages.

An application object itself would be enough to create non-GUI programs with just ARexx and commodities capabilities. If you want to have windows with lots of nice gadgets and other user interface stuff, you will have to add window objects to your application. Since the application object is able to to handle any number of children, the number of windows is not limited.

Window objects are instances of window class and handle all the actions related with opening, closing, moving, resizing and refreshing of intuition windows. However, a window for itself is not of much use without having any contents to display. That's why window objects always need a so called root object.

With this root object, we finally reach the gadget related classes of the MUI system. These gadget related classes are all subclasses of Area class, they describe a rectangle region with some class dependent contents. Many different classes such as strings, buttons, checkmarks or listviews are available, but the most important subclass of area class is probably the Group class. Instances of this class are able to handle any number of child objects and control the size and position of these children with various attributes. Of course these children can again be group objects with other sets of children. Since you usually want your window to contain more than just one object, the root object of a window must always be a group class object.

In MUI Royale GUIs are defined entirely using the XML markup language. Here is an example what an application tree could look like in an XML declaration:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application base="HELLOWORLD">
   <window title="Example GUI" muiid="MAIN">
      <vgroup>
         <listview>
            <column/>
         </listview>
         <string/>
         <hgroup>
            <button>Add</button>
            <button>Remove</button>
         </hgroup>
      </vgroup>
   </window>
</application>
```

Because these first paragraphs are very important for understanding how MUI works, here's a brief summary:

An application consists of exactly one application object. This application object may have any number of children, each of them being a window object. Every window object contains a root object that must be of type group class. This group object again handles any number of child objects, either other group objects or some user interface elements such as strings, sliders or buttons.

3.2 Class hierarchy

There are two important super classes that you have to know because almost all other MUI classes are children of these two super classes: The first one is Notify class which handles MUI's event notification system. Notify class allows your MUI objects to listen to certain events and trigger notifications whenever these events occur. See Section 30.1 [Notify class], page 175, for details.

The second important super class is Area class. Area class is the the super class for all MUI objects that have a visual representation in form of a gadget in a MUI window. Thus, you can use all of Area class' attributes on all of your MUI gadgets. For example, you can use the `Area.Width` and `Area.Height` attributes to find out the dimensions of a listview, texteditor, or string gadget because these are all subclasses of Area class. See Section 7.1 [Area class], page 55, for details.

3.3 Automatic layout engine

One of the most important and powerful features of MUI is its dynamic layout engine. As opposed to other available user interface tools, the programmer of a MUI application doesn't have to care about gadget sizes and positions. MUI handles all necessary calculations automatically, making every program completely screen, window size and font sensitive without the need for the slightest programmer interaction.

From a programmer's point of view, all you have to do is to define some rectangle areas that shall contain the objects you want to see in your window. Objects of group class are used for this purpose. These objects are not visible themselves, but instead tell their children whether they should appear horizontally or vertically.

For automatic and dynamic layout, it's important that every single object knows about its minimum and maximum dimensions. Before opening a window, MUI asks all its gadgets about these values and uses them to calculate the window's extreme sizes.

Once the window is opened, layout takes place. Starting with the current window size, the root object and all its children are placed depending on the type of their father's group and on some additional attributes. The algorithm ensures that objects will never become smaller as their minimum or larger as their maximum size.

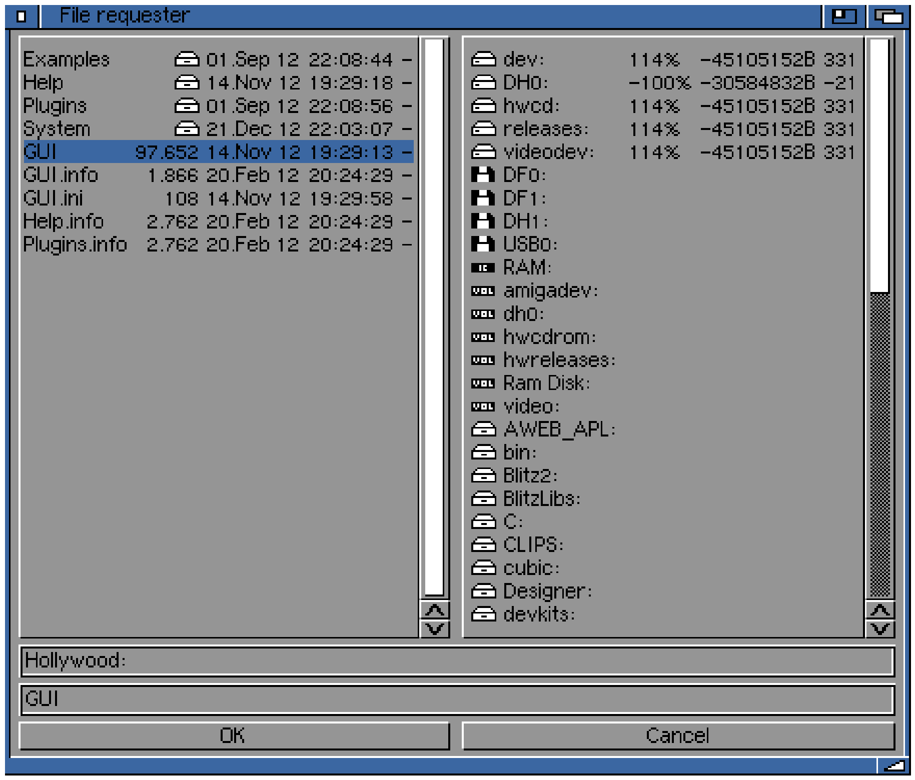
The important thing with this mechanism is that object placement depends on window size. This allows very easy implementation of a sizing gadget: whenever the user resizes a window, MUI simply starts a new layout process and recalculates object positions and sizes automatically. No programmer interaction is needed.

As a consequence, you should never explicitly define fixed sizes for your windows and your windows' gadgets. Instead, MUI calculates these automatically depending on the current user preferences. Although it is possible to use fixed sizes for your window using `Window.Width` and `Window.Height` it is against MUI's philosophy to actually use these.

Also, the gadget counterparts `Area.FixWidth` and `Area.FixHeight` should only be used if you have a very good reason for doing so. Normally, you should leave all layout calculation as well as gadget and window size determination to MUI's automatic layout engine which will make sure to give the user the ideal visual representation of your GUI layout.

3.4 Group objects

As mentioned above, a programmer specifies a window's design by grouping objects either horizontally or vertically. As a little example, let's have a look at a simple file requester window:



```
Examples         01.Sep 12 22:08:44 -
Help             14.Nov 12 19:29:18 -
Plugins          01.Sep 12 22:08:56 -
System           21.Dec 12 22:03:07 -
GUI       97.652 14.Nov 12 19:29:13 -
GUI.info   1.866 20.Feb 12 20:24:29 -
GUI.ini      108 14.Nov 12 19:29:58 -
Help.info  2.762 20.Feb 12 20:24:29 -
Plugins.info 2.762 20.Feb 12 20:24:29 -
```

```
dev:          114%  -45105152B 331
DH0:         -100%  -30584832B -21
hwcd:         114%  -45105152B 331
releases:     114%  -45105152B 331
videodev:     114%  -45105152B 331
DF0:
DF1:
DH1:
USB0:
RAM:
amigadev:
dh0:
hwcdrom:
hwreleases:
Ram Disk:
video:
AWEB_APL:
bin:
Blitz2:
BlitzLibs:
C:
CLIPS:
cubic:
Designer:
devkits:
```

Hollywood:

GUI

This window consists of two listview objects, two string gadgets and two buttons. To tell MUI how these objects shall be placed, you need to define groups around them. Here, the window consists of a vertical group that contains a horizontal group with both lists as first child, the path gadget as second child, the file gadget as third child and again a horizontal group with both buttons as fourth child.

In an XML file, the GUI definition of this window could then look like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application base="FILEREQ">
   <window title="File requester">
      <vgroup>
         <hgroup>
            <listview><column/></listview>
            <listview><column/></listview>
```

```
                </hgroup>
                <string/>
                <string/>
                <hgroup>
                   <button>OK</button>
                   <button>Cancel</button>
                </hgroup>
             </vgroup>
          </window>
       </application>
```

This XML definition is completely enough to define the contents of our window, all necessary sizes and positions are automatically calculated by the MUI system.

To understand how these calculations work, it's important to know that all basic objects (e.g. strings, buttons, lists) have a fixed minimum and a maximum size. Group objects calculate their minimum and maximum sizes from their children, depending whether they are horizontal or vertical:

Horizontal groups:

- The minimum width of a horizontal group is the sum of all minimum widths of its children.
- The maximum width of a horizontal group is the sum of all maximum widths of its children.
- The minimum height of a horizontal group is the biggest minimum height of its children.
- The maximum height of a horizontal group is the smallest maximum height of its children.

Vertical groups:

- The minimum height of a vertical group is the sum of all minimum heights of its children.
- The maximum height of a vertical group is the sum of all maximum heights of its children.
- The minimum width of a vertical group is the biggest minimum width of its children.
- The maximum width of a vertical group is the smallest maximum width of its children.

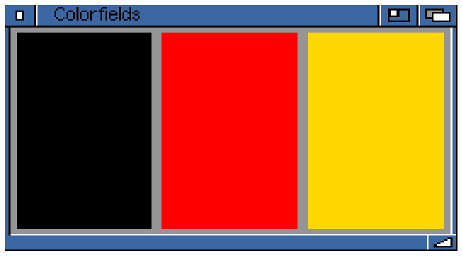
Maybe this algorithm sounds a little complicated, but in fact it is really straight forward and ensures that objects will neither get smaller as their minimum nor bigger as their maximum size.

Before a window is opened, it asks its root object (usually a group object) to calculate minimum and maximum sizes. These sizes are used as the window's bounding dimensions, the smallest possible window size will result in all objects being display in their minimum size.

Once minimum and maximum sizes are calculated, layout process starts. The root object is told to place itself in the rectangle defined by the current window size. This window size is either specified by the programmer or results from a window resize operation by the user. When an object is told to layout itself, it simply sets its position and dimensions to the given rectangle. In case of a group object, a more or less complicated algorithm distributes all available space between its children and tells them to layout too.

This "more or less complicated algorithm" is responsible for the object arrangement. Depending on some attributes of the group object (horizontal or vertical, ...) and on some attributes of the children (minimum and maximum dimensions, ...), space is distributed and children are placed.

A little example makes things more clear. Let's see what happens in a window that contains nothing but three horizontally grouped colorfield objects:



Colorfield objects have a minimum width and height of one pixel and no maximum width and height. Since we have a horizontal group, the minmax calculation explained above yields to a minimum width of three pixels and a minimum height of one pixel for the window's root object (the horizontal group containing the colorfields). Maximum dimensions of the group are unlimited. Using these results, MUI is able to calculate the windows bounding dimensions by adding some spacing values and window border thicknesses.

Once min and max dimensions are calculated, the window can be opened with a programmer or user specified size. This size is the starting point for the following layout calculations. For our little example, let's imagine that the current window size is 100 pixels wide and 50 pixels high.

MUI subtracts the window borders and some window inner spacing and tells the root object to layout itself into the rectangle left=5, top=20, width=90, height=74. Since our root object is a horizontal group in this case, it knows that each colorfield can get the full height of 74 pixels and that the available width of 90 pixels needs to be shared by all three fields. Thus, the resulting fields will all get a width of 90/3=30 pixels.

That's the basic way MUI's layout system works. There are a lot more possibilities to influence layout, you can e.g. assign different weights to objects, define some inter object spacing or even make two-dimensional groups. These sophisticated layout issues are discussed in the documentation of group class.

3.5 Object handling

MUI Royale uses XML files to create MUI objects. When creating objects you can use all attributes marked with the letter "I" in the applicability section of the accompanying attribute documentation. For example, to create a string object with a string-kind frame, a maximum length of 80 and the initial contents "foobar", you would have to use the following XML code:

```
<string id="mystring" frame="string" contents="foobar" maxlen="80"/>
```

Once your object is ready, you can start talking to it by setting or getting one of its attributes or by sending it methods. For that purpose it is important that you give your object an identifier using the "id" tag so that the functions `mui.Set()`, `mui.Get()` and `mui.DoMethod()` can find your object. Here's an example of talking to your string gadget:

```
mui.Set("mystring", "contents", "look")
s$ = mui.Get("mystring", "contents")
DebugPrint("Always " .. s$ .. " on the bright side of life.")
```

As already mentioned above, all attributes and methods are completely documented in the documentation coming with this distribution.

3.6 Notifications

The central element for controlling a MUI application is the notification mechanism. To understand how it works, it's important to know that most objects feature lots of attributes that define their current state. Notification makes it possible to react on changes of these attributes.

Attributes are changed either directly by the programmer (with a call to `mui.Set()`) or by the user manipulation of some gadgets. If he e.g. changes the active entry of a cycle gadget, the `Cycle.Active` attribute will continously be updated and reflect the currently active entry.

If you want to be informed whenever the value of a certain object attribute changes, you first have to setup a notification for this attribute in your object declaration. This is done directly in the XML file by using the "Notify" tag:

```
<cycle id="mycycle" notify="active">
   <item>One</item>
   <item>Two</item>
   <item>Three</item>
</cycle>
```

If you want to listen to multiple notifications on the same MUI object, you have to separate them using semicolons, e.g.:

```
<cycle id="mycycle" notify="active; appmessage">
...
</cycle>
```

Note that you can also setup or remove notifications at run-time using the `mui.Notify()` function from your code.

The next thing you have to do is setup an event handler for events from MUI Royale using the `InstallEventHandler()` Hollywood function. You have to pass a Hollywood function that shall act as an event handler to the `InstallEventHandler()` function. Here is an example:

```
InstallEventHandler({MUIRoyale = p_MUIHandler})
```

Whenever a MUI event occurs, MUI Royale will then call the event handler callback function that you passed to `InstallEventHandler()`. The function will receive a table as its parameter with the following fields initialized:

`Action:` Initialized to "MUIRoyale".

`Class:` Contains the name of the MUI class this event comes from, e.g. "Cycle".

`Attribute:`
Contains the name of the class attribute that has triggered the event, e.g. "Active".

`ID:` Contains the ID of the MUI object that triggered this event, e.g. "mycycle".

`TriggerValue:`
Contains the current value of the attribute that triggered this event. You could also find this out by doing a `mui.Get()` on the object but it is more convenient to get the current value directly to your event callback.

`MUIUserData:`
If the object was assigned certain userdata, it will be passed to the event handler callback in this tag. See Section 30.12 [Notify.UserData], page 179, for details.

`NotifyData:`
If the object was assigned certain notify data, it will be passed to the event handler callback in this tag. See Section 30.10 [Notify.NotifyData], page 178, for details.

3.7 Applicability

In the documentation of every object attribute you will find information about the applicability of this attribute. Attribute applicability is described in the form of a combination of the four letters I, S, G, and N. This tells you the various contexts that the attribute can be used in.

Here is an explanation of the different applicability contexts:

`I` Attribute can be used when creating the object in the XML file. (initialization time)

`S` Attribute can be used with `mui.Set()` at runtime.

`G` Attribute can be used with `mui.Get()` at runtime.

`N` Notifications on this attribute are possible either by using the "Notify" tag in the XML declaration or by calling the `mui.Notify()` function at runtime.

For example, if an attribute has an applicability of just "I", then this attribute can only be used during object initialization time. It cannot be changed later using `mui.Set()`. If an attribute has an applicability of just "S" on the other hand, it is not possible to specify the attribute already at initialization time in the XML file. Attributes that have an applicability of "ISGN" can be used in all contexts.

3.8 Cycle chain

One of the big advantages of MUI is that it also offers powerful features to control your GUI via keyboard. For this purpose you should setup a keyboard cycle chain for all your windows. Whenever the user presses the TAB key then, the next MUI object in the keyboard cycle chain gets activated.

All you have to do to add a MUI object to the keyboard cycle chain is setting its `Area.CycleChain` attribute to `True`. For example, to add a listview to the keyboard cycle chain, just do the following in your XML declaration:

```
<listview cyclechain="true">
   <column/>
</listview>
```

Objects of type Button class do not need to be added to the keyboard cycle chain explicitly. As a convenience feature, MUI Royale will add all buttons to the keyboard cycle chain automatically. If you do not want this, you have to set `Area.CycleChain` to `False` for all your buttons, although there is really no sound reason for doing that.

Please make sure that you always add all your MUI objects to the keyboard cycle chain because this makes your GUI much more attractive for keyboard-only users.

3.9 Text formatting codes

MUI's text engine is very powerful and allows you to include special character codes for on-the-fly text formatting. For example, it is possible to specify text alignment and font style (bold, italics, etc.) using formatting codes. With MUI 3.9 or better it is even possible to embed images (from a Hollywood brush source) in your text objects.

Formatting codes always start with an escape character followed by a sequence of characters that describe the formatting code. In decimal notation the escape character equals ASCII code 27, which is 33 in octal and $1B in hexadecimal notation. Care has to be taken when using formatting codes because usage is different between XML files and Hollywood source files. In XML files the octal notation is used, i.e. you start an escape sequence using a backslash and the octal number 33 ('\33'). In Hollywood source codes, however, octal numbers are not supported after a backslash. Hollywood always expects the ASCII code in decimal notation after a backslash. That is why you have to use '\27' to initiate an escape sequence from Hollywood code.

To illustrate this difference a little bit better, let us have a look at two examples. Here is an example for creating a bold text object in an XML file. Bold text is enabled by using the character 'b' after the escape character:

```
<text id="mytext">\33bBold text</text>
```

You can see that the octal notation is used here because XML files expect an octal number after a backslash. In Hollywood, however, it is different because Hollywood expects a decimal character after a backslash. So here is how you have to specify escape codes when using them from a Hollywood source file:

```
mui.Set("mytext", "contents", "\27bBold text")
```

You can see that the code is the same except that we use \27b instead of \33b because Hollywood always uses decimal instead of octal numbers after a backslash.

The following formatting codes are currently supported:

`\33u`	Set the soft style to underline.
`\33b`	Set the soft style to bold.
`\33i`	Set the soft style to italic.

`\33n` Set the soft style back to normal.

`\33<n>` Use pen number n (2..9) as front pen. n must be a valid DrawInfo pen as specified in `intuition/screens.h`.

`\33P[RRGGBB]`
: Change front color to the specified RGB color. The RGB color has to be specified in the form of six hexadecimal digits RRGGBB. This is only possible on true colour screens. It also requires at least MUI 3.9.

`\33P[AARRGGBB]`
: Change front color to the specified RGB color and apply alpha blending at the specified intensity. The RGB color has to be specified in the form of six hexadecimal digits RRGGBB which have to be prefaced with two alpha channel digits AA specifying the blending intensity. This is only possible on true colour screens. It also requires at least MUI 3.9. Due to limitations in Picasso96 this does not work with AmigaOS 4 yet (as of AmigaOS 4.1 Update 6).

`\33c` Center current (and following) line(s). This sequence is only valid at the beginning of a string or after a newline character.

`\33r` Right justify current (and following) line(s). This sequence is only valid at the beginning of a string or after a newline character.

`\33l` Left justify current (and following) line(s). This sequence is only valid at the beginning of a string or after a newline character.

`\33A[s]` Include the Hollywood brush that uses the identifier `<s>` in the text object. This feature requires at least MUI 3.9.

`\33-` Disable text engine, following chars will be printed without further parsing.

`\n` Start a new line. With this character you can e.g. create multi line buttons.

Please note: These formatting codes can be used in all MUI strings, not only in text objects. You can e.g. format the columns of a listview or include images in a cycle gadget's entries.

3.10 Implementing online help

MUI provides several ways of implementing direct and immediate online help functionality in your applications. First of all, every MUI gadget can have a bubble help functionality that pops up after the mouse has hovered over a gadget for a user-specified interval. This is done via the `Area.ShortHelp` attribute. As Area class is the super class for all MUI gadgets, you can use this attribute to add a help text to all your gadgets. Here is an example:

```
<vgroup>
   <listview shorthelp="List of loaded video files">
      <column/>
   </listview>
   <button shorthelp="Plays a video stream">Play</button>
   <button shorthelp="Stops a video stream">Stop</button>
   <button shorthelp="Exits the program.">Quit</button>
</vgroup>
```

Furthermore, you can specify an external AmigaGuide file for your application that is opened whenever the user presses the HELP key. This is done by setting the `Application.HelpFile` attribute. Here is an example:

```
<application helpfile="PROGDIR:MyProgram.guide">
...
</application>
```

To make your online help functionality even more convenient for the user, you can fine-tune the access to the external AmigaGuide file by specifying which node or line of the AmigaGuide file should be displayed when the mouse is over a specific object. This is done by using the `Notify.HelpNode` or the `Notify.HelpLine` attributes. For example, let's assume that the node "VideoPlayback" should be displayed when the mouse is over the "Play" button and the user presses the HELP key. You can write this as the following XML code:

```
<button helpnode="VideoPlayback">Play</button>
```

Of course, your application has to define `Application.HelpFile` first if you want to use `Notify.HelpNode`.

If your help system is not that detailed that you have an own node for every gadget and you just have an own node for every window, then you can also use `Notify.HelpNode` on window level. Like this:

```
<window helpnode="VideoPlayback">
...
</window>
```

Whenever that window is the active one then and the user presses HELP, MUI will automatically open the AmigaGuide file specified in `Application.HelpFile` and display the node specified in `Notify.HelpNode`. You see that MUI makes it really easy for the programmers to implement a convenient help system in their applications. All you have to do is use it for your own application and your end-users will surely be grateful for it.

3.11 Context menus

In MUI GUIs every gadget can have its own context menu that is automatically popped up whenever the mouse pointer is over the gadget and the user holds down the right mouse button. Context menus are assigned to the different gadgets by using the `Area.ContextMenu` attribute. As Area class is the super-class for all MUI gadgets, you can use this attribute with every MUI object that has a visual representation in form of a gadget.

`Area.ContextMenu` expects a MUI menustrip object as its argument so you have to create a menustrip for your context menu in XML first. It is very important to note that you have to declare your menustrips in the `<application>` scope because menustrips are global objects and are only attached to windows or gadgets later on. That is why it is not allowed to declare menustrips inside a `<window>` XML scope.

Here is an example in which we add a cut, copy, and paste context menu to an object of type Texteditor class:

```
<menustrip id="ctxtmenu">
   <menu title="Context menu">
      <item>Cut</item>
```

```
            <item>Copy</item>
            <item>Paste</item>
        </menu>
    </menustrip>

    <window>
    ...
        <texteditor contextmenu="ctxtmenu"/>
    ...
    </window>
```

Please note that menustrips which are used as context menus must not contain more than one `<menu>` tree.

3.12 Drag'n'drop

Unfortunately, MUI Royale currently does not support all of MUI's drag'n'drop functionality. It is, however, possible to use drag'n'drop operations that come from Workbench in the form of app messages.

If your window should be able to accept these messages, you first have to set the `Window.AppWindow` attribute to `True`. After that you have to define which of your MUI gadgets should accept dropped files. This is done by setting up a notification on the `Notify.AppMessage` attribute for the desired gadgets. As Notify class is the super-class for all MUI gadgets, you can use this attribute with every MUI object that has a visual representation in form of a gadget. If you do not want to have specific gadgets as drop targets, you can also setup your whole window as a drop target by simply setting up a notification on the `Notify.AppMessage` attribute for your whole window.

Whenever the user drops an icon over your window or MUI gadget then, you will get a notification of type `AppMessage`. See Section 30.2 [Notify.AppMessage], page 175, for details.

When your application is iconified and the user drops files on your app icon you can use the attribute `Application.DropObject` to define the MUI object which should receive the dropped file(s).

3.13 Character encoding

MUI Royale supports two character encodings in the XML files used to describe the GUI layout: `iso-8859-1` and `utf-8`. If you use `utf-8` encoding, MUI Royale will require `codesets.library` to do character conversion from UTF-8 to the system's default charset.

Please note that MUI itself does not support UTF-8. MUI always uses the system's default charset. Thus, if you use UTF-8 encoding in your XML files, MUI Royale will try to map these strings to the system's default charset. This may not always succeed. For example, if the system's default charset is ISO-8859-1 and the UTF-8 XML file uses some Eastern European characters not present in ISO-8859-1 then they will not be displayed correctly. They will only be displayed correctly if the system's default charset has them as well.

3.14 Hollywood bridge

A powerful feature of MUI Royale is that it allows you to embed complete Hollywood displays inside your MUI GUIs using Hollywood class. Whenever you draw something to a Hollywood display that is attached to Hollywood class, it will automatically be drawn to your MUI GUI as well. You can even hide the Hollywood display so that the user does not even notice that Hollywood is running in the background. Furthermore, all mouse clicks and key strokes that happen inside Hollywood class will be forwarded to the corresponding Hollywood display as normal Hollywood events. Thus, Hollywood class allows you to use almost all of Hollywood's powerful features inside a MUI GUI as well.

Let's have a look at an example. The following code uses Hollywood class to embed a playing animation of size 320x200 inside a MUI GUI. For this, the Hollywood display's size is changed to 320x200 and then it is embedded in the MUI GUI using a MUI object of type Hollywood class. Here is the XML GUI declaration first:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application base="HOLIBRIDGE">
   <window title="Hollywood bridge" muiid="MAIN" notify="closerequest">
      <vgroup>
         <hgroup>
            <rectangle/>
            <hollywood display="1"/>
            <rectangle/>
         </hgroup>
         <hgroup>
            <button id="play" notify="pressed">Play</button>
            <button id="stop" notify="pressed">Stop</button>
         </hgroup>
      </vgroup>
   </window>
</application>
```

Note that we use MUI objects of Rectangle class to pad the non-resizable Hollywood object. This is necessary for the GUI to stay resizable. Here is the code now that shows you to connect MUI and Hollywood:

```
@ANIM 1, "amy_walks.anim"
@DISPLAY {Width = 320, Height = 200, Hidden = True}

Function p_AnimFunc()
   Local numframes = GetAttribute(#ANIM, 1, #ATTRNUMFRAMES)
   curframe = Wrap(curframe + 1, 1, numframes + 1)
   DisplayAnimFrame(1, 0, 0, curframe)
EndFunction

Function p_EventFunc(msg)
   Switch msg.Class
   Case "Window":
      Switch msg.Attribute
      Case "CloseRequest":
         End
      EndSwitch
   Case "Button":
      Switch msg.Attribute
      Case "Pressed":
         Switch msg.ID
         Case "play":
            SetInterval(1, p_AnimFunc, 50)
         Case "stop":
            ClearInterval(1)
         EndSwitch
      EndSwitch
   EndSwitch
EndFunction

InstallEventHandler({MUIRoyale = p_EventFunc})
mui.CreateGUI(FileToString("GUI.xml"))

Repeat
   WaitEvent
Forever
```

With a little bit more work we can also make the anim movable with the mouse. The user can then click into the Hollywood object and drag the anim around using his mouse. Here is the code for that:

```
@ANIM 1, "amy_walks.anim"
@DISPLAY {Width = 320, Height = 200, Hidden = True}

Function p_AnimFunc()
   Local numframes = GetAttribute(#ANIM, 1, #ATTRNUMFRAMES)
   curframe = Wrap(curframe + 1, 1, numframes + 1)
   SelectBrush(1)
   Cls
   DisplayAnimFrame(1, offx, offy, curframe)
   EndSelect
   DisplayBrush(1, 0, 0)
EndFunction

Function p_MouseFunc()
   If IsLeftMouse() = True
     Local mx, my = MouseX(), MouseY()
     If (grabx = -1) And (graby = -1) Then
       grabx, graby = mx - offx, my - offy
     offx = mx - grabx
     offy = my - graby
   Else
```

```
      grabx, graby = -1, -1
   EndIf
EndFunction

Function p_EventFunc(msg)
   Switch msg.Class
   Case "Window":
      Switch msg.Attribute
      Case "CloseRequest":
         End
      EndSwitch
   Case "Button":
      Switch msg.Attribute
      Case "Pressed":
         Switch msg.ID
         Case "play":
            SetInterval(1, p_AnimFunc, 50)
            SetInterval(2, p_MouseFunc, 20)
         Case "stop":
            ClearInterval(1)
            ClearInterval(2)
         EndSwitch
      EndSwitch
   EndSwitch
EndFunction

CreateBrush(1, 320, 200)

InstallEventHandler({MUIRoyale = p_EventFunc})
mui.CreateGUI(FileToString("GUI8.xml"))

Repeat
   WaitEvent
Forever
```

Finally, it is also possible to make the Hollywood object resizable by specifying the attributes `Hollywood.MinWidth`, `Hollywood.MinHeight`, `Hollywood.MaxWidth`, and `Hollywood.MaxHeight`. Whenever the user resizes the MUI window, your Hollywood display will receive a `SizeWindow` event that you can listen to using the `InstallEventHandler()` Hollywood function. When using a resizable Hollywood object, we can remove the two `<rectangle>` objects used solely as padding space. The XML code looks like this then:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application base="HOLIBRIDGE">
   <window title="Hollywood bridge" muiid="MAIN" notify="closerequest">
      <vgroup>
         <hollywood display="1" minwidth="32" minheight="32"
```

```
             maxwidth="16384" maxheight="16384"/>
          <hgroup>
             <button id="play" notify="pressed">Play</button>
             <button id="stop" notify="pressed">Stop</button>
          </hgroup>
       </vgroup>
    </window>
 </application>
```

Note that we specify 16384 for both `Hollywood.MaxWidth` and `Hollywood.MaxHeight`. This is to indicate that our Hollywood object doesn't really have a maximum size. That is why we set these two attribute to the maximum allowable size of 16384 pixels. This should be sufficiently large enough for most purposes.

Our code is pretty much the same as before with the exception that we now have to handle the `SizeWindow` event to take care of GUI resize events. Here is the adapted code from above:

```
@ANIM 1, "amy_walks.anim"
@DISPLAY {Width = 320, Height = 200, Hidden = True}

Function p_AnimFunc()
   Local numframes = GetAttribute(#ANIM, 1, #ATTRNUMFRAMES)
   curframe = Wrap(curframe + 1, 1, numframes + 1)
   SelectBrush(1)
   Cls
   DisplayAnimFrame(1, offx, offy, curframe,
     {Width = swidth, Height = sheight})
   EndSelect
   DisplayBrush(1, 0, 0)
EndFunction

Function p_MouseFunc()
   If IsLeftMouse() = True
     Local mx, my = MouseX(), MouseY()
     If (grabx = -1) And (graby = -1) Then
       grabx, graby = mx - offx, my - offy
     offx = mx - grabx
     offy = my - graby
   Else
     grabx, graby = -1, -1
   EndIf
EndFunction

Function p_EventFunc(msg)
   If msg.Action = "SizeWindow"
     swidth = msg.Width
     sheight = msg.Height
     CreateBrush(1, swidth, sheight)
```

```
      Return
   EndIf

   Switch msg.Class
   Case "Window":
      Switch msg.Attribute
      Case "CloseRequest":
         End
      EndSwitch
   Case "Button":
      Switch msg.Attribute
      Case "Pressed":
         Switch msg.ID
         Case "play":
            SetInterval(1, p_AnimFunc, 50)
            SetInterval(2, p_MouseFunc, 20)
         Case "stop":
            ClearInterval(1)
            ClearInterval(2)
         EndSwitch
      EndSwitch
   EndSwitch
EndFunction

swidth, sheight = 320, 200
CreateBrush(1, swidth, sheight)

InstallEventHandler({MUIRoyale = p_EventFunc, SizeWindow = p_EventFunc})
mui.CreateGUI(FileToString("GUI8.xml"))

Repeat
   WaitEvent
Forever
```

From this example you can see that Hollywood class is really quite powerful and can be used to achieve lots of innovative GUI ideas only limited by your creativity. By the way, the example above is also included in the MUI Royale distribution. You can find it in the drawer `Examples/Hollywood`.

3.15 Style guide

Note: These topics aren't discussed here just for fun. You will annoy lots of users if you don't pay attention to them!

File Requester:

Even if MUI features a file list and a volume list object and makes building a private file requester very easy, you should always provide a possibility to pop up a standard asl requester for this purpose. Just add a little popup button right beneath your file string gadget and everything will be fine. MUI offers a file-popup object exactly for this purpose. Note well: Many users (including myself) move programs with non-standard file requesters into the trashcan immediately.

Window Size:
: With MUI, it's very easy to have lots of gadgets within a single window. Since you as a programmer usually have a more powerful system with higher graphic resolutions as most of your users, windows tend to become too big. You should always make sure that everything you design fits on a standard 640x256 screen with a topaz/8 font. Otherwise, MUI will try to use very small fonts or virtual groups to make your window fit, making your application look and feel bad.

Keyboard Control:
: Even if you're a "mouse-only" user, add keyboard cycle chains and gadget shortcuts to your application. It's very few work for you and helps lots of users. See Section 3.8 [Cycle chain], page 13, for details.

Background:
: MUI allows the user to adjust lots of different backgrounds for objects. Even if you don't use this feature, you should always test your program with a fancy background pattern configuration and check whether all your buttons really have button backgrounds, all your framed texts really have text backgrounds, etc.

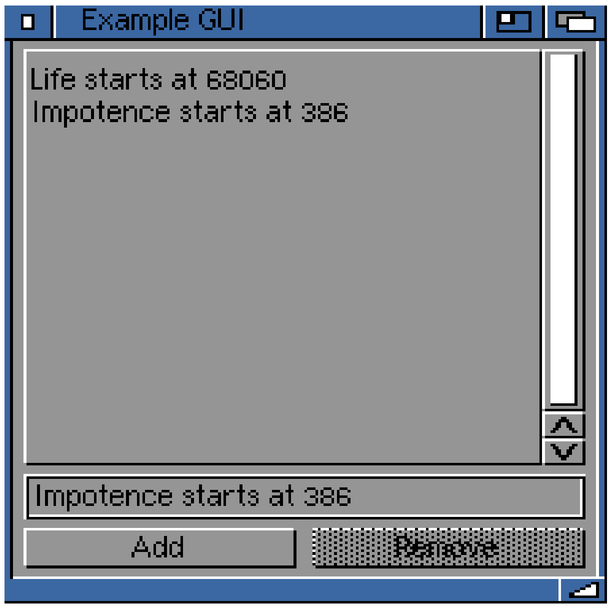
Add a shortcut to MUI preferences:
: Every MUI program should include a menu item that allows the user to configure the appearance of the program using the MUI preferences program. This can be done very easily by adding a menu item that simply runs the method `Application.OpenConfigWindow` on the application object. Also, you should include "About MUI" and "About MUI Royale" menu items that show the "About MUI" and "About MUI Royale" windows which you can open using the `Application.AboutMUI` and `Application.AboutMUIRoyale` methods respectively.

And last... Don't forget the traditional Amiga style guide!

4 Tutorial

4.1 Tutorial

Welcome to the MUI Royale tutorial! This small step-by-step document will guide you through the process of creating your first GUI with MUI Royale in very few steps.



Life starts at 68060
Impotence starts at 386

Impotence starts at 386

Let us start with the basics: In MUI Royale GUIs are created using XML files that contain a description of a number of windows containing a variety of GUI elements. Here is a minimal GUI description for a MUI Royale GUI that contains a window with a listview, a string gadget, and two buttons in XML format:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application base="HELLOWORLD">
   <window title="Example GUI" muiid="MAIN">
      <vgroup>
         <listview>
            <column/>
         </listview>
         <string/>
         <hgroup>
            <button>Add</button>
            <button>Remove</button>
         </hgroup>
      </vgroup>
   </window>
</application>
```

The application base is set to `HELLOWORLD`. It is very important that you always define the `Application.Base` attribute because the string you specify in this attribute is used by

MUI to store the program-specific preferences in an external file (usually in `ENV:MUI`). The `<application>` object is the MUI master object for every application and must only be used once per application. All other MUI objects are children of Application class.

It is also important to set the attribute `Window.MuiID` for every window because only windows that have a MUI id remember their position and size. Also, only windows that have `Window.MuiID` can be snapshot by the user from the window's context menu. Thus, it is very important that you set this attribute for all your windows.

To see how our XML declaration above looks as a MUI GUI, we have to save it to a file named `GUI.xml` and then use the following code to turn it into a full-blown MUI GUI:

```
@DISPLAY {Hidden = True}

mui.CreateGUI(FileToString("GUI.xml"))

Repeat
   WaitEvent
Forever
```

Please note that we use the `@DISPLAY` preprocessor command to hide the Hollywood display that will popup by default. We do not want to have this display because we already have a MUI GUI that shall be the visual representation of our program. Keep in mind, though, that the Hollywood display is still there. It is just hidden. Hollywood's conception says that there always must be a valid display. That is why we cannot remove the Hollywood display completely. All we can do is to hide it. You can still draw to it, though.

You should also add some information about your program using the `@APPAUTHOR`, `@APPCOPYRIGHT`, `@APPDESCRIPTION`, `@APPTITLE`, and `@APPVERSION` preprocessor commands. MUI needs this information for several purposes, e.g. the information passed in `@APPTITLE` is used by the MUI preferences window. Here is an example declaration of these preprocessor commands:

```
@APPTITLE "Tutorial"
@APPVERSION "$VER: Tutorial 1.0 (27.12.12)"
@APPCOPYRIGHT "Copyright ©2012, Andreas Falkenhahn"
@APPAUTHOR "Andreas Falkenhahn"
@APPDESCRIPTION "The tutorial app from the MUI Royale guide"
```

When you run the code above you will notice that the window does not react on clicks on its close gadget. You have to use CTRL-C or Exchange to close the program. Of course, we want our program to be closable via the window's close gadget as well. In order to achieve this, we need to setup a notification on the `Window.CloseRequest` attribute that will be set whenever the user hits the close gadget. We can setup this notification by modifying our XML code from above like this:

```
<window title="Example GUI" muiid="MAIN" notify="closerequest">
```

The next thing we have to do is install an event handler callback using the `InstallEventHandler()` Hollywood function because our Hollywood script needs to be informed every time a MUI Royale event comes in. Thus, we have to modify our code as follows:

```
@DISPLAY {Hidden = True}
```

```
Function p_EventFunc(msg)

   Switch msg.Class
   Case "Window":
      Switch msg.Attribute
      Case "CloseRequest":
         End
      EndSwitch
   EndSwitch

EndFunction

InstallEventHandler({MUIRoyale = p_EventFunc})
mui.CreateGUI(FileToString("GUI.xml"))

Repeat
   WaitEvent
Forever
```

What we have done here is installing the function `p_EventFunc` as an event handler callback that gets executed whenever a MUI Royale event comes in. When we get such an event, we then have to check which MUI class and attribute has triggered it. This is done by looking into the `msg.Class` and `msg.Attribute` fields of the event message that our callback receives as its first parameter. In order to react on the `Window.CloseRequest` attribute, we thus have to look for an event from class `Window` and attribute `CloseRequest`. When we have found such an event, we simply call the `End()` command to terminate our program.

The next thing we want to do is add the functionality that whenever the user presses the "Add" button the text in the string gadget should get added to the listview as the last entry. To do this, we first have to find a way of identifying our MUI gadgets from the Hollywood script. This is done by giving them IDs in the XML declaration. IDs are simply text strings that are used for talking to MUI objects from Hollywood scripts. So let's add some IDs now for all gadgets that we need to talk to. We have to modify our XML declaration like this:

```
...
         <listview id="mylistview">
            <column/>
         </listview>
         <string id="mystring"/>
         <hgroup>
            <button id="mybt1">Add</button>
            <button id="mybt2">Remove</button>
         </hgroup>
...
```

Now we have to setup some notifications so that our event handler callback does not only inform us about changes in the `Window.CloseRequest` attribute but also in the `Button.Pressed` attribute which is the attribute that gets set to `True` whenever the user hits a button. To listen to these button clicks we have to modify our code like this:

```
...
            <button id="mybt1" notify="pressed">Add</button>
            <button id="mybt2" notify="pressed">Remove</button>
...
```

Now that we have done this we can add some code to our event handler callback that grabs the contents of the string gadget and adds it to the end of the list in our listview object. This is done by first calling `mui.Get()` on the `String.Contents` attribute to get the contents of the string gadget and then running the method `Listview.Insert` on the listview gadget using `mui.DoMethod()` to insert the entry into the listview. Here is the code that has to be inserted into `p_EventFunc` for this purpose:

```
Switch msg.Class
...
Case "Button":
   Switch msg.Attribute
   Case "Pressed":
      Switch msg.ID
      Case "mybt1":    ; "Add" button was pressed
         Local s$ = mui.Get("mystring", "contents")
         mui.DoMethod("mylistview", "insert", "bottom", s$)
      EndSwitch
   EndSwitch
EndSwitch
```

The next thing we want to do is implement the functionality of our "Remove" button. Whenever this button is pressed, we want the active entry to be removed from the listview. We can do this by running the `Listview.Remove` method on the listview. Hence, we have to modify our code like this:

```
      Switch msg.ID
      ...
      Case "mybt2":    ; "Remove" button was pressed
         mui.DoMethod("mylistview", "remove", "active")
      EndSwitch
```

Now we want the active entry of the listview to be automatically displayed in the string gadget. For this purpose we have to setup a notification on the `Listview.Active` attribute which is triggered whenever the active entry of the listview changes. Thus, we have to modify our XML file like this:

```
...
         <listview id="mylistview" notify="active">
            <column/>
         </listview>
...
```

In our event handler callback we can implement this functionality quite easily by running the `Listview.GetEntry` method and then setting the string gadgets contents using the `String.Contents` attribute. Here is the code for doing that:

```
Switch msg.Class
...
```

```
    Case "Listview":
       Switch msg.Attribute
       Case "Active":
          Local s$ = mui.DoMethod("mylistview", "getentry", "active")
          mui.Set("mystring", "contents", s$)
       EndSwitch
    EndSwitch
```

If you try this code, you will set that the `Listview.Active` attribute is not only triggered when the user selects a new listview entry with his mouse, but also when entries are removed from the listview and thus cause a new entry becoming the active one.

The next thing we want to do is disable the "Remove" button when there is no active entry in the listview. We can disable MUI gadgets by setting the `Area.Disabled` attribute to `True`. As there are no entries in the listview initially, we have to set `Area.Disabled` to `True` already at the start of our program. So you have to insert this code:

```
...
mui.CreateGUI(FileToString("GUI.xml"))
mui.Set("mybt2", "disabled", True)
...
```

Now we have to make some modifications to our event handler callback. Whenever we get the notification on `Listview.Active` we have to check if it is different from the special value "Off". If that is the case, we will enable the "Remove" gadget. The special value "Off" (-1) is returned by `Listview.Active` whenever there is no active entry in the listview. We have to modify our code like this:

```
    Switch msg.Class
    ...
    Case "Listview":
       Switch msg.Attribute
       Case "Active":
          Local s$ = mui.DoMethod("mylistview", "getentry", "active")
          mui.Set("mystring", "contents", s$)
          mui.Set("mybt2", "disabled", IIf(msg.triggervalue = -1,
              True, False))
       EndSwitch
    EndSwitch
```

We use the field `msg.TriggerValue` here. This always contains the current value of the attribute that has triggered the event, i.e. in our case it contains the current value of the `Listview.Active` attribute. We could also call `mui.Get()` manually on `Listview.Active` first, but this is not really required because we can simply use the `msg.TriggerValue` shortcut.

The last thing we want to do is add a menu to our GUI. All MUI programs should have a menu item that allows the user to customize the program's appearance using the MUI preferences. This is done by running the `Application.OpenConfigWindow` method on the application object. Also, every MUI program should have the menu items "About MUI" and "About MUI Royale" that inform the user that this program was done using MUI and MUI Royale. You can popup the these windows by running the `Application.AboutMUI` and `Application.AboutMUIRoyale` methods on the application object respectively. To add these menus to our program, we use the Menustrip class. Here is the XML code that you have to add before your window declaration:

```
...
<application base="HELLOWORLD">
   <menustrip id="mymenustrip">
      <menu title="Project">
         <item id="menabout" notify="selected">About...</item>
         <item id="menaboutmui" notify="selected">About MUI...</item>
         <item id="menaboutmuiroyale" notify="selected">
            About MUI Royale...</item>
         <item/>
         <item id="menquit" notify="selected" shortcut="Q">Quit</item>
      </menu>
      <menu title="Settings">
         <item id="menmuiset" notify="selected">MUI...</item>
      </menu>
   </menustrip>
   ...
</application>
```

After we have created our menustrip object using the XML code above we have to attach this menustrip to our window. This is done by setting the `Window.Menustrip` attribute to our menustrip object. Here is the XML code for this:

```
<window title="Example GUI" muiid="MAIN" notify="closerequest"
    menustrip="mymenustrip">
```

As you can see in the XML code above, we have already added notifications on the `Menuitem.Selected` attribute to get informed whenever the user selects a menu item. Now we have to add code to our event handler function that takes the appropriate action when a menu item is selected. Before we can do that, however, we need to assign an ID to our application object because we need to use `mui.DoMethod()` on it. Here is how the XML code needs to be adapted:

```
<application base="HELLOWORLD" id="app">
```

Now we can write the code for our event handler callback function that handles menu items:

```
Switch msg.Class
...
Case "Menuitem":
   Switch msg.Attribute
   Case "Selected":
      Switch msg.id
      Case "menabout":
         mui.Request("Test", "Test program\n" ..
             "© 2012 by Andreas Falkenhahn", "OK")
      Case "menaboutmui":
         mui.DoMethod("app", "aboutmui")
      Case "menaboutmuiroyale":
```

```
                mui.DoMethod("app", "aboutmuiroyale")
             Case "menquit":
                End
             Case "menmuiset":
                mui.DoMethod("app", "openconfigwindow")
             EndSwitch
          EndSwitch
       EndSwitch
```

Some final touches to our program could be adding online help to our gadgets by using the `Area.ShortHelp` attribute and adding all objects to the keyboard cycle chain so that the GUI can also be controlled via keyboard. We do not have to add the buttons to our keyboard cycle chain, because this is always done automatically. We have to add the listview and the string gadgets manually, though. Adding objects to the keyboard cycle chain is done by using the `Area.CycleChain` attribute. Also, it is advised that you listen to the `ShowWindow` and `HideWindow` events that can come from the Exchange program in Commodities and act accordingly. So here is how our final program looks like with these two last enhancements applied. First the XML file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application base="HELLOWORLD" id="app">
   <menustrip id="mymenustrip">
      <menu title="Project">
         <item id="menabout" notify="selected">About...</item>
         <item id="menaboutmui" notify="selected">About MUI...</item>
         <item id="menaboutmuiroyale" notify="selected">
            About MUI Royale...</item>
         <item/>
         <item id="menquit" notify="selected" shortcut="Q">Quit</item>
      </menu>
      <menu title="Settings">
         <item id="menmuiset" notify="selected">MUI...</item>
      </menu>
   </menustrip>
   <window title="Example GUI" muiid="MAIN" notify="closerequest"
             menustrip="mymenustrip">
      <vgroup>
        <listview id="mylistview" notify="active" shorthelp="A listview"
            cyclechain="true">
            <column/>
         </listview>
         <string id="mystring" shorthelp="Enter entry name here"
             cyclechain="true"/>
         <hgroup>
            <button id="mybt1" notify="pressed"
                  shorthelp="Add new entry">Add</button>
            <button id="mybt2" notify="pressed"
                  shorthelp="Remove entry">Remove</button>
         </hgroup>
      </vgroup>
   </window>
</application>
```

And here is the code for the program logic:

```
@DISPLAY {Hidden = True}

@APPTITLE "Tutorial"
@APPVERSION "$VER: Tutorial 1.0 (27.12.12)"
@APPCOPYRIGHT "Copyright ©2012, Andreas Falkenhahn"
@APPAUTHOR "Andreas Falkenhahn"
@APPDESCRIPTION "The tutorial app from the MUI Royale guide"

Function p_EventFunc(msg)

   If msg.Action <> "MUIRoyale"

      Switch msg.Action
      Case "HideWindow":
         mui.Set("app", "iconified", True)
      Case "ShowWindow":
         mui.Set("app", "iconified", False)
      EndSwitch

      Return

   EndIf

   Switch msg.Class
   Case "Window":
      Switch msg.Attribute
      Case "CloseRequest":
         End
      EndSwitch

   Case "Button":
      Switch msg.Attribute
      Case "Pressed":
         Switch msg.ID
         Case "mybt1":    ; "Add" button was pressed
            Local s$ = mui.Get("mystring", "contents")
            mui.DoMethod("mylistview", "insert", "bottom", s$)
         Case "mybt2":    ; "Remove" button was pressed
            mui.DoMethod("mylistview", "remove", "active")
         EndSwitch
      EndSwitch
```

```
    Case "Listview":
       Switch msg.Attribute
       Case "Active":
          Local s$ = mui.DoMethod("mylistview", "getentry", "active")
          mui.Set("mystring", "contents", s$)
          mui.Set("mybt2", "disabled", IIf(msg.triggervalue = -1,
              True, False))
       EndSwitch

    Case "Menuitem":
       Switch msg.Attribute
       Case "Selected":
          Switch msg.id
          Case "menabout":
             mui.Request("Test", "Test program\n" ..
                 "© 2012 by Andreas Falkenhahn", "OK")
          Case "menaboutmui":
             mui.DoMethod("app", "aboutmui")
          Case "menaboutmuiroyale":
             mui.DoMethod("app", "aboutmuiroyale")
          Case "menquit":
             End
          Case "menmuiset":
             mui.DoMethod("app", "openconfigwindow")
          EndSwitch
       EndSwitch
    EndSwitch

EndFunction

InstallEventHandler({MUIRoyale = p_EventFunc, HideWindow = p_EventFunc,
    ShowWindow = p_EventFunc})
mui.CreateGUI(FileToString("GUI.xml"))
mui.Set("mybt2", "disabled", True)

Repeat
    WaitEvent
Forever
```

That's it! Now you should be able to create MUI programs to your personal taste. Thank you for reading this tutorial and enjoy the power of MUI with Hollywood and MUI Royale at your hands!

5 Function reference

5.1 mui.CreateGUI

NAME

mui.CreateGUI – create application object from an XML source

SYNOPSIS

`mui.CreateGUI(xml$)`

FUNCTION

This function creates a MUI GUI from the XML description passed in the xml$ argument, i.e. it establishes the MUI master application object with all of its children. Please note that xml$ must be a string that contains the XML GUI declaration and not a filename. If you want to use an XML GUI declaration from an external file, you have to convert that file into a string first, e.g. using the `FileToString()` Hollywood function.

Once this function returns, you can talk to all your MUI objects that you have defined in the XML GUI declaration using the `mui.Set()`, `mui.Get()` and `mui.DoMethod()` functions.

Please note that there can be only one application object per task so this function can only be called once. If you want to call `mui.CreateGUI()` a second time, you have to free the old GUI first using the `mui.FreeGUI()` call.

If you need to dynamically add objects like windows or buttons to your application object, you can use the `mui.CreateObject()` function to do this.

INPUTS

`xml$`	a string containing an XML GUI description

EXAMPLE

```
mui.CreateGUI([[
<?xml version="1.0" encoding="iso-8859-1"?>
<application>
   <window title="Test program" notify="closerequest">
      <vgroup>
         <button>Hello World!</button>
      </vgroup>
   </window>
</application>
]])

InstallEventHandler({MUIRoyale = Function(msg)
      If msg.attribute = "CloseRequest" Then End
   EndFunction})

Repeat
   WaitEvent
Forever
```

The code above creates a minimal GUI, just with a window and a single button.

5.2 mui.CreateObject

NAME

mui.CreateObject – create MUI object from an XML source (V1.2)

SYNOPSIS

`mui.CreateObject(xml$)`

FUNCTION

This function can be used to dynamically create a MUI object from an XML source. When `mui.CreateObject()` returns, the newly created MUI object won't be attached to any parent object and will live in a state of isolation from your application object created by `mui.CreateGUI()`. Thus, to break this state of isolation you first have to attach the object to a parent object which can be either a group object, a menu object, or an application object. If your newly allocated MUI object is a window object, you will have to to attach it to the application object by using the `Application.AddWindow` method. To attach menu objects you have to use the `Menustrip.AddHead`, `Menustrip.AddTail`, `Menustrip.Insert`, `Menu.AddHead`, `Menu.AddTail`, or `Menu.Insert` methods. All other objects can be attached by using the group methods `Group.AddHead`, `Group.AddTail` and `Group.Insert`,

In contrast to `mui.CreateGUI()` you can call `mui.CreateObject()` as often as you like as it doesn't create an application object for you but just detached MUI objects of which you can have as many as you like.

It is important that you specify an ID for your MUI object in the XML declaration because you need this ID to refer to this object when you want to add it to an application or group object.

Once this function returns, you can talk to the newly created MUI object (and to all of its children) using the `mui.Set()`, `mui.Get()` and `mui.DoMethod()` functions.

Detached MUI objects can be freed using the `mui.FreeObject()` function but you only have to call this in specific cases, e.g. if you are dealing with lots of dynamically allocated MUI objects and you want to do some housekeeping to save on memory and resources.

INPUTS

`xml$`	a string containing an XML MUI object description

EXAMPLE

```
mui.CreateObject([[
<window id="newwindow" title="A new window" notify="closerequest">
  <vgroup>
      <button>Hello World!</button>
   </vgroup>
</window>
]])

mui.DoMethod("app", "addwindow", "newwindow")
mui.Set("newwindow", "open", True)
```

The code above creates a new window, adds it to the existing application object and opens it.

```
mui.CreateObject([[
<button id="newbutton">Dynamically created button!</button>
]])

mui.DoMethod("mygroup", "initchange")
mui.DoMethod("mygroup", "addtail", "newbutton")
mui.DoMethod("mygroup", "exitchange", false)
```

The code above dynamically creates a new button object and adds it as the last child to the group that has the ID "mygroup".

5.3 mui.DoMethod

NAME

mui.DoMethod – run method on MUI object

SYNOPSIS

```
r = mui.DoMethod(id$, method$, ...)
```

FUNCTION

This function can be used to run a method on the specified MUI object. You have to pass the identifier of the MUI object in the first argument and the name of the method in the second argument. Method and object IDs are case insensitive, i.e. it does not matter if you use upper or lower case characters.

The methods that you can use with this function depend on the class of the specified MUI object. Have a look at the class reference to see what methods are supported by the different MUI classes.

Also, the arguments that you have to pass to this function after the method name depend on the method. They are different for every method. The same is true for return values. Some methods return values, some do not. Please refer to the class reference to see which arguments your method requires and whether there are return values for your method.

INPUTS

`id$`	identifier of MUI object to run method on
`method$`	method name as a string
`...`	additional arguments depend on the method (see class reference for details)

RESULTS

`r`	return value depends on method type

EXAMPLE

```
mui.DoMethod("my_listview", "insert", "bottom", "Last entry")
```

The code above adds a new entry named "Last entry" to the bottom of the listview using the identifier "my_listview". This is done by running the `Listview.Insert()` method on the listview object.

5.4 mui.FreeGUI

NAME

mui.FreeGUI – delete entire application object

SYNOPSIS

`mui.FreeGUI()`

FUNCTION

Use this call to delete the entire GUI object created using the last call to `mui.CreateGUI()`. `mui.FreeGUI()` will delete the whole MUI master application object. All windows will be closed and their children will be deleted. After this call returns, you could create a new GUI using the `mui.CreateGUI()` function if you want.

INPUTS

none

5.5 mui.FreeImage

NAME

mui.FreeImage – free brush in image cache (V1.7)

SYNOPSIS

`mui.FreeImage(id)`

FUNCTION

This function can be used to free a brush in MUI Royale's internal image cache. You have to pass the identifier of the brush that should be freed. Alternatively, you can also pass -1 to this function to free all brushes in MUI Royale's internal image cache.

You must make sure that the specified brush is no longer used by any widgets in your GUI before you call this function. Note that under normal conditions it is not necessary to call this function because normally all brushes are freed automatically by MUI Royale. Under certain conditions, however, it can be useful to call this function.

MUI Royale caches all Hollywood brushes that you use as images in your GUI. That is why when you try to use a Hollywood brush in your GUI a second time, it will just be loaded from MUI Royale's internal image cache for performance reasons, regardless of the brush's current contents inside Hollywood. This can lead to unwanted behaviour in case you have updated your brush's graphics in the meantime and you want MUI Royale to use the updated graphics. In that case, you first have to free the brush in MUI Royale's internal image cache by using this function. When you pass the brush to MUI Royale again then, it will be re-created from the brush's current state and it will be cached anew.

INPUTS

`id`	identifier of brush to free or -1 to free all brushes

5.6 mui.FreeObject

NAME

mui.FreeObject – delete a detached MUI object (V1.2)

SYNOPSIS

```
mui.FreeObject(id$)
```

FUNCTION

This function can be used to delete a detached MUI object that has been created either by `mui.CreateObject()` or `mui.CreateGUI()`. The MUI object that you specify here must not be attached to an application, group, or menu object any longer because attached MUI objects are freed with their parent so make sure you only use this function with MUI objects that have been detached from their parent and are no longer bound to any parent object.

To detach MUI objects from their respective parents, you have to use one of the following methods: `Application.RemoveWindow`, `Menustrip.Remove`, `Menu.Remove` or `Group.Remove`.

When MUI Royale exits, it will automatically free all detached MUI objects so unless your program constantly adds and removes MUI objects at runtime, you will normally not have to call this function at all.

INPUTS

`id$`	identifier of MUI object to free

5.7 mui.Get

NAME

mui.Get – get value of a MUI object attribute

SYNOPSIS

```
r = mui.Get(id$, attr$)
```

FUNCTION

This function can be used to retrieve the current value of the attribute attr$ in the MUI object specified in id$. Attribute names and object IDs are case insensitive, i.e. it does not matter if you use upper or lower case characters for them.

The attributes that you can use with this function depend on the class of the specified MUI object. Have a look at the class reference to see what attributes are supported by the different MUI classes. In order to use an attribute with this function, it needs to have an applicability of `"G"`. Attributes of Area class and Notify class can be used on almost all other classes because the Area and Notify classes act as superclasses for most of the other classes.

INPUTS

`id$`	identifier of MUI object to query
`attr$`	attribute whose value should be retrieved

RESULTS

`r`	current attribute value

EXAMPLE

```
DebugPrint(mui.Get("my_listview", "active"))
```

The code above returns the index of the currently active entry in the listview that has the identifier "my_listview" by querying the Listview.Active attribute.

5.8 mui.HaveObject

NAME

mui.HaveObject – check if MUI object exists (V1.7)

SYNOPSIS

```
r = mui.HaveObject(id$)
```

FUNCTION

This function simply checks whether a MUI object of the given `id$` exists or not. If it exists, `True` is returned, `False` otherwise.

INPUTS

`id$`	MUI object id to check

RESULTS

`r`	`True` or `False` depending on whether the object exists

5.9 mui.IsVersion4

NAME

mui.IsVersion4 – check if MUI 4 or better is installed (V1.2)

SYNOPSIS

```
r = mui.IsVersion4()
```

FUNCTION

This function can be used to check whether the user is running MUI 4.0 and up. As some features of MUI Royale are only available on MUI 4, it might be necessary to use this function to check whether they are available before you try to use them.

INPUTS

none

RESULTS

`r`	returns `True` if MUI 4 or better is available

5.10 mui.Notify

NAME

mui.Notify – add/remove notification on a MUI object attribute

SYNOPSIS

```
mui.Notify(id$, attr$, enable)
```

FUNCTION

This function can be used to add or remove a notification on the attribute attr$ in the MUI object specified in id$. You have to pass the attribute name and a boolean flag that indicates whether you want to enable or disable notifications on that attribute. Attribute names and object IDs are case insensitive, i.e. it does not matter if you use upper or lower case characters for them.

The attributes that you can use with this function depend on the class of the specified MUI object. Have a look at the class reference to see what attributes are supported by the different MUI classes. In order to use an attribute with this function, it needs to have an applicability of "N". Attributes of Area class and Notify class can be used on almost all other classes because the Area and Notify classes act as superclasses for most of the other classes.

Once you have setup a notification on a certain object attribute, you can listen to these events by installing a MUI Royale event handler callback using the `InstallEventHandler()` Hollywood function. See Section 3.6 [Notifications], page 12, for details.

Please note that notifications can also be setup in the XML GUI declaration by using the `Notify` tag. See Section 3.6 [Notifications], page 12, for details.

INPUTS

`id$`	identifier of MUI object to use
`attr$`	attribute to listen to
`enable`	`True` to add a notification or `False` to remove notification from this object

EXAMPLE

```
mui.Notify("my_listview", "active", True)
```

The code above installs a notification that triggers whenever the Listview.Active attribute changes in the listview that has the identifier "my_listview".

5.11 mui.Request

NAME

mui.Request – popup a MUI system requester

SYNOPSIS

```
r = mui.Request(title$, body$, gadgets$[, icon$])
```

FUNCTION

This function pops up a standard MUI requester that displays a message (body$) and also allows the user to make a selection using one of the gadgets specified by gadgets$.

Using a MUI requester instead of a standard system requester offers you the possibility to include text containing all the text engine format codes.

Separate the gadgets specified in gadgets$ by a "|". The return value tells you which gadget the user pressed. Please note that the right most gadget always has the value of `False` (0) because it is typically used as the "Cancel" gadget. If you have for example three gadget choices "One|Two|Three", the gadget "Three" has return value 0, "Two" returns 2, and "One" returns 1.

If you precede an entry with a '*' in gadgets$, this answer will become the active object. Pressing <Return> will terminate the requester with this response. A '_' character indicates the keyboard shortcut for this response.

The strings you specify in body$ and gadgets$ can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

Starting with MUI Royale 1.4 this function accepts an optional argument named icon$. This allows you to specify the icon to show in the requester window. Note that this is currently only supported by MUI 4 and up on AmigaOS 3 and AmigaOS 4. The MorphOS MUI doesn't support this feature yet. The following predefined values are accepted for icon$:

`"None":`	no icon
`"Information":`	an information sign
`"Error":`	an error sign
`"Warning":`	a warning sign
`"Question":`	a question mark

INPUTS

`title$`	title for the requester; pass an empty string ("") to use the default title
`body$`	text to appear in the body of the requester
`gadgets$`	one or more gadgets that the user can press
`icon$`	optional: icon to show in the requester (defaults to "Information") (V1.4)

RESULTS

`r`	the gadget that was pressed by the user

EXAMPLE

```
mui.Request("MUI Royale", "\027c\027b\027uHello!\n\n" ..
    "\027nDo you like MUI Royale!", "*_Yes|_No")
```

The code above demonstrates the use of the `mui.Request()` function.

5.12 mui.Set

NAME

mui.Set – set value of a MUI object attribute

SYNOPSIS

```
mui.Set(id$, attr1$, val1$, ...)
```

FUNCTION

This function can be used to set the current value of one or more attributes in the MUI object specified in id$. You have to pass the attribute name and desired new value for every attribute you want to modify. You can repeat these attribute/value pairs as often as you like to modify multiple attributes with just a single call to `mui.Set()`. Attribute names and object IDs are case insensitive, i.e. it does not matter if you use upper or lower case characters for them.

The attributes that you can use with this function depend on the class of the specified MUI object. Have a look at the class reference to see what attributes are supported by the different MUI classes. In order to use an attribute with this function, it needs to have an applicability of "S". Attributes of Area class and Notify class can be used on almost all other classes because the Area and Notify classes act as superclasses for most of the other classes.

If you have setup a notification on the attribute that you want to modify using this function, the notification will be triggered once you call `mui.Set()` on that attribute. If you do not want this behaviour, you can use the `Notify.NoNotify` attribute to prevent a notification from being issued.

INPUTS

`id$`	identifier of MUI object to modify
`attr1$`	attribute whose value should be modified
`val1$`	new value for the attribute
`...`	you can repeat attribute/value pairs as often as you like

EXAMPLE

```
mui.Set("my_listview", "active", 15)
```

The code above sets entry number 15 as currently active entry in the listview that has the identifier "my_listview" by setting the Listview.Active attribute.

```
mui.Set("my_listview", "nonotify", True, "active", 15)
```

This code does the same as the code above but prevents notifications from being issued by setting the `Notify.NoNotify` attribute to `True`. This is useful if you need to distinguish between user selections in the listview and selections made programmatically using `mui.Set()`.

6 Application class

6.1 Overview

Application class is the master class for all MUI applications. It serves as a kind of anchor for all input, either coming from the user or somewhere from the system, e.g. commodities or ARexx messages.

An application can have any number of sub windows, these windows are the children of the application. Every GUI definition has to start with the application class.

6.2 Application.AboutMUI

NAME

Application.AboutMUI – show the MUI about window

SYNOPSIS

```
mui.DoMethod(id, "AboutMUI")
```

FUNCTION

Show the MUI about window. Please include that in all your applications and link with a menu item called "About MUI...".

INPUTS

`id`	id of the application object

6.3 Application.AboutMUIRoyale

NAME

Application.AboutMUIRoyale – show the MUI Royale about window

SYNOPSIS

```
mui.DoMethod(id, "AboutMUIRoyale")
```

FUNCTION

Show the MUI Royale about window. Please include that in all your applications and link with a menu item called "About MUI Royale...".

INPUTS

`id`	id of the application object

6.4 Application.AddWindow

NAME

Application.AddWindow – add detached window object to application object (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddWindow", window)
```

FUNCTION

This method can be used to add a detached window object to the application object. Once the window object has been attached to the application object, you can open the window by setting its `Window.Open` attribute.

Detached window objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Application.RemoveWindow` method.

INPUTS

`id`	id of the application object
`window`	id of the window object to add

EXAMPLE

See Section 5.2 [mui.CreateObject], page 36.

6.5 Application.Base

NAME

Application.Base – set/get basename of an application

FUNCTION

The basename for an application. This name is used for the builtin ARexx port and for some internal file management.

A basename must neither contain spaces nor any special characters such as ":/()#?*...".

When your program is a single task application (i.e. `Application.SingleTask` is `True`), the base name will be used without further modification.

Otherwise, it gets a ".1", ".2", etc. appended, depending on how many applications are already running. If you need to know the name of your ARexx port, you can query the base name attribute after the application is created.

TYPE

String

APPLICABILITY

IG

6.6 Application.ConfigChange

NAME

Application.ConfigChange – get notified about configuration changes (V1.2)

FUNCTION

This attribute will trigger whenever the user changes the configuration of your application. Be warned that this can trigger quite often because it will be set for every single thing the user changes.

This is only supported on MUI 4.x.

TYPE

Boolean

APPLICABILITY

N

6.7 Application.DoubleStart

NAME

Application.DoubleStart – get notified about double start of application

FUNCTION

This attribute is set automatically when the user tries to start an application that has the attribute `Application.SingleTask` set twice. You can react on this and take appropriate actions, e.g. pop up a requester or quit yourself.

TYPE

Boolean

APPLICABILITY

GN

6.8 Application.DropObject

NAME

Application.DropObject – set default drop object

FUNCTION

If your application is iconified and the user drops icons onto the AppIcon, the object specified here will receive the `Notify.AppMessage` notification.

TYPE

String

APPLICABILITY

IS

6.9 Application.HelpFile

NAME

Application.HelpFile – set/get help file of application

FUNCTION

This attribute allows defining an AmigaGuide style file to be displayed when the user requests online help.

When the HELP button is pressed and the application defines a `Application.HelpFile`, MUI tries to obtain `Notify.HelpNode` from the current object (the one under the mouse pointer). If `Notify.HelpNode` is not defined, MUI continues asking the parent object for this attribute (usually a group, but remember: the parent of a windows root object is the window itself, the parent of a window is the application).

When an `Notify.HelpNode` is found, the same procedure is applied to `Notify.HelpLine`. Then MUI puts the application to sleep and displays the file at the position specified with `Notify.HelpNode` and/or `Notify.HelpLine`.

This behaviour allows you to define one `Application.HelpFile` for your application object and different help nodes and lines for your applications windows and/or gadgets.

See Section 3.10 [Implementing online help], page 15, for details.

TYPE

String

APPLICABILITY

ISG

6.10 Application.Icon

NAME

Application.Icon – set application icon

FUNCTION

If you specify the path to an icon file here, MUI will use this object for the AppIcon when your application gets iconified. Otherwise MUI will try to locate `env:sys/dev_mui.info` and, if not present, fall back to a default icon.

TYPE

String

APPLICABILITY

I

6.11 Application.Iconified

NAME

Application.Iconified – (un)iconify application

FUNCTION

Setting this attribute to `True` causes the application to become iconified. Every open window will be closed and a (configurable) AppIcon will appear on the workbench.

Same thing happens when the user hits the iconify gadget in the window border or uses commodities Exchange to hide your applications interface.

When an application is iconified and you try to open a window, the window won't open immediately. Instead MUI remembers this action and opens the window once the application is uniconified again.

TYPE

Boolean

APPLICABILITY

SG

6.12 Application.Load

NAME

Application.Load – load program configuration (V1.7)

SYNOPSIS

`mui.DoMethod(id, "Load", name$)`

FUNCTION

`Application.Save`, `Application.Load` and `Notify.ExportID` offer an easy way of saving and loading a program's configuration.

Each gadget with a `Notify.ExportID` will get its contents saved during `Application.Save` and restored during `Application.Load`. This makes it very easy to design a configuration window with "Save", "Use" and "Cancel" buttons to allow the user storing the settings. When the application starts, you would just have to call `Application.Load` and the stored settings will be read and installed.

The `name$` parameter must be set to the name of the file you wish to load the settings from. Usually you won't need to think of a real name but instead use one of the following special values:

`::ENV`	Load settings from ENV:.
`::ENVARC`	Load settings from ENVARC:.

Not all classes are able to import and export their contents. Currently, you may define a `Notify.ExportID` for

`String class`
: `String.Contents` is ex/imported.

`Radio class`
: `Radio.Active` is ex/imported.

`Cycle class`
: `Cycle.Active` is ex/imported.

`Listview class`
: `Listview.Active` is /ex/imported.

`Text class`
: `Text.Contents` is ex/imported.

`Slider class`
: `Slider.Level` is ex/imported.

`Area class`
: `Area.Selected` is ex/imported (e.g. for Checkmark gadgets)

`Menuitem class`
: `Menuitem.Selected` is ex/imported.

`Group class`	`Group.ActivePage` is ex/imported.

INPUTS

`id`	id of the application object
`name$`	name of the file you wish to load the settings from or special value (see above)

6.13 Application.Menustrip

NAME

Application.Menustrip – set menustrip for the entire application

FUNCTION

Specify a menu strip object for the application. Menustrip objects defined for the application are used as menu for every window of the application, as long as the window doesn't define its private menu.

You have to pass the identifier of a menustrip from the `Menustrip` MUI class here.

TYPE

MUI object

APPLICABILITY

I

6.14 Application.OpenConfigWindow

NAME

Application.OpenConfigWindow – show MUI preferences window

SYNOPSIS

```
mui.DoMethod(id, "OpenConfigWindow")
```

FUNCTION

Since MUI 3, applications can open their own MUI configuration window to allow users to adjust the local preferences without the need of an external program. Programmers are supposed to include a "Settings/MUI..." menu item which simply calls `Application.OpenConfigWindow`. MUI will then automatically show the preferences window without blocking the rest of the program.

INPUTS

`id`	id of the application object

6.15 Application.RemoveWindow

NAME

Application.RemoveWindow – detach window object from application object (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "RemoveWindow", window)
```

FUNCTION

This method removes the specified window object from the application object and puts the window object in detached state. Window objects in detached state can either be reattached by running the `Application.AddWindow` method or can be freed by calling `mui.FreeObject()`.

INPUTS

`id`	id of the application object
`window`	id of the window object to remove

6.16 Application.Save

NAME

Application.Save – save program configuration (V1.7)

SYNOPSIS

```
mui.DoMethod(id, "Save", name$)
```

FUNCTION

`Application.Save`, `Application.Load` and `Notify.ExportID` offer an easy way of saving and loading a program's configuration.

Each gadget with a `Notify.ExportID` will get its contents saved during `Application.Save` and restored during `Application.Load`. This makes it very easy to design a configuration window with "Save", "Use" and "Cancel" buttons to allow the user storing the settings. When the application starts, you would just have to call `Application.Load` and the stored settings will be read and installed.

The `name$` parameter must be set to the name of the file you wish to save the settings to. Usually you won't need to think of a real name but instead use one of the following special values:

`::ENV`	Save settings to ENV:.
`::ENVARC`	Save settings to ENVARC:.

Not all classes are able to import and export their contents. Currently, you may define a `Notify.ExportID` for

`String class`
: `String.Contents` is ex/imported.

`Radio class`
: `Radio.Active` is ex/imported.

`Cycle class`
: `Cycle.Active` is ex/imported.

`Listview class`
: `Listview.Active` is /ex/imported.

`Text class`
: `Text.Contents` is ex/imported.

`Slider class`
: `Slider.Level` is ex/imported.

`Area class`
: `Area.Selected` is ex/imported (e.g. for Checkmark gadgets)

`Menuitem class`
: `Menuitem.Selected` is ex/imported.

`Group class`
: `Group.ActivePage` is ex/imported.

INPUTS

`id`	id of the application object
`name$`	name of the file you wish to save the settings to or special value (see above)

6.17 Application.SingleTask

NAME

Application.SingleTask – set operation mode of application

FUNCTION

Boolean value to indicate whether or not your application is a single task program. When set to `True`, MUI will refuse to create more than one application object.

In this case, the already running application gets its `Application.DoubleStart` attribute set to `True`. You can listen to this and take appropriate actions, e.g. pop up a requester.

Examples for single task applications are the system preferences program. It doesn't make sense for them to run more than once.

TYPE

Boolean

APPLICABILITY

I

6.18 Application.Sleep

NAME

Application.Sleep – put application to sleep

FUNCTION

This attribute can be used to put a whole application to sleep. All open windows get disabled and a busy pointer appears.

This attribute contains a nesting count, if you tell your application to sleep twice, you will have to tell it to wake up twice too.

If you need to do some time consuming actions, you always should set this attribute to inform the user that you are currently unable to handle input.

A sleeping application's windows cannot be resized.

TYPE

Boolean

APPLICABILITY

S

6.19 Application.WindowList

NAME

Application.WindowList – get a list of all application windows (V1.2)

FUNCTION

This attribute allows you to get a table that contains the identifiers of all windows that are currently attached to the application object.

TYPE

Table

APPLICABILITY

G

7 Area class

7.1 Overview

Area class is a super class for every other MUI class except windows and applications. It holds information about an object's current position, size and weight and manages frames, fonts and backgrounds.

Because Area class is the super class for every MUI gadget, you can use all of its attributes with all other MUI classes that create gadgets. For example, you can use `Area.ShortHelp` to add bubble help to a button, or you could use `Area.ContextMenu` to add a context menu to a listview, etc.

7.2 Area.Background

NAME

Area.Background – adjust background of an object

FUNCTION

Adjust the background for an object.

Every MUI object has its own background setting. The background is displayed "behind" the actual object contents, e.g. behind a the text of a text object or behind the image of an image object.

An object without a specific background setting will inherit the pattern from its parent group. The default background for a window and many other background patterns are adjustable with the preferences program.

The following background settings are possible:

- WindowBack
- RequesterBack
- ButtonBack
- ListBack
- TextBack
- PropBack
- PopupBack
- SelectedBack
- RegisterBack
- GroupBack
- SliderBack
- PageBack
- ReadListBack
- pre_Background
- pre_Shadow
- pre_Shine

- pre_Fill
- pre_ShadowBack
- pre_ShadowFill
- pre_ShadowShine
- pre_FillBack
- pre_FillShine
- pre_ShineBack
- pre_FillBack2
- pre_HshineBack
- pre_HshadowBack
- pre_HshineShine
- pre_HshadowShadow
- pre_MarkShine
- pre_MarkHalfshine
- pre_MarkBackground

All backgrounds with "pre" are MUI's predefined patterns. These are not configurable by the user and will always look the same.

Note: It is *important* that you test your programs with a fancy pattern configuration. With the default setting you won't notice any errors in your backgrounds.

TYPE

String (see above for possible values)

APPLICABILITY

IS

7.3 Area.BackgroundBrush

NAME

Area.BackgroundBrush – set background to a MUI brush (V1.5)

FUNCTION

Uses the specified MUI brush as the background for the object. You must pass the name of an external MUI brush to this attribute.

Every MUI object has its own background setting. The background is displayed "behind" the actual object contents, e.g. behind a the text of a text object or behind the image of an image object.

An object without a specific background setting will inherit the pattern from its parent group. The default background for a window and many other background patterns are adjustable with the preferences program.

Please note that it is generally not a good idea to change the background to a hard-coded image because this can easily clash with the user's current skin settings. Remember that everything in MUI is user-configurable so you should not override these settings by defining your own skins.

TYPE

String

APPLICABILITY

IS

7.4 Area.BackgroundImage

NAME

Area.BackgroundImage – set background to an image (V1.2)

FUNCTION

Uses the specified image as the background for the object. You must pass a filename to an image in this attribute. The image is then loaded via datatypes.

Every MUI object has its own background setting. The background is displayed "behind" the actual object contents, e.g. behind a the text of a text object or behind the image of an image object.

An object without a specific background setting will inherit the pattern from its parent group. The default background for a window and many other background patterns are adjustable with the preferences program.

Please note that it is generally not a good idea to change the background to a hard-coded image because this can easily clash with the user's current skin settings. Remember that everything in MUI is user-configurable so you should not override these settings by defining your own skins.

TYPE

String

APPLICABILITY

IS

7.5 Area.BackgroundRGB

NAME

Area.BackgroundRGB – set background to an RGB color (V1.2)

FUNCTION

Sets the background of this object to the specified RGB color.

Every MUI object has its own background setting. The background is displayed "behind" the actual object contents, e.g. behind a the text of a text object or behind the image of an image object.

An object without a specific background setting will inherit the pattern from its parent group. The default background for a window and many other background patterns are adjustable with the preferences program.

Please note that it is generally not a good idea to change the background to a hard-coded color value because this can easily clash with the user's current skin settings. Remember that everything in MUI is user-configurable so you should not override these settings by defining your own color schemes.

From the Hollywood script the color is specified as a simple numerical value containing 8 bits for each component. When you specify the color in the XML file, it has to be passed as a 6 character string prefixed by the #-character (just like in HTML).

TYPE

Number

APPLICABILITY

IS

7.6 Area.BottomEdge

NAME

Area.BottomEdge – get bottom edge of object

FUNCTION

You can use this to read the current position and dimension of an object, if you e.g. need it to pop up some requester below.

Of course, this attribute is only valid when the parent window of the object is currently open.

TYPE

Number

APPLICABILITY

G

7.7 Area.ContextMenu

NAME

Area.ContextMenu – set context menu for object

FUNCTION

Specifies a context sensitive popup menu for the current object. For MUI, popup menus are nothing else but standard intuition menus, so you must specify the id of a MUI menustrip object here.

Whenever the user hits the RMB and the mouse is above the parent object, MUI will present the popup menu instead of the windows menu.

Please note that menustrips which are used as context menus must not contain more than one `<menu>` tree.

In MUI Royale 1.0 this attribute could only be used at initialization stage. Starting with MUI Royale 1.1 it has an applicability of "IS" which means that you can also change context menus at runtime. To remove a context menu completely, pass the special string "(none)" to this attribute.

See Section 29.1 [Menustrip class], page 171, for an example.

TYPE

MUI object

APPLICABILITY

IS

7.8 Area.ContextMenuTrigger

NAME

Area.ContextMenuTrigger – react on context menu events

FUNCTION

This notification will return the id of the context menu item that the user has selected.

TYPE

String

APPLICABILITY

N

7.9 Area.ControlChar

NAME

Area.ControlChar – set shortcut for object

FUNCTION

Pressing the control char will have the same effect as pressing return if the object was active.

This can be used to create old style key shortcuts.

Note: Using an uppercase control char will force the user to press shift.

TYPE

Single character string

APPLICABILITY

ISG

7.10 Area.CycleChain

NAME

Area.CycleChain – set/get cycle chain flag

FUNCTION

MUI 3 introduces a new keyboard cycle chain system. All you have to do is to set `Area.CycleChain` to `True` for every object that you want to have in your chain, MUI does the rest automatically.

See Section 3.8 [Cycle chain], page 13, for details.

TYPE
Boolean

APPLICABILITY
ISG

7.11 Area.Disabled

NAME
Area.Disabled – set/get disabled state

FUNCTION
Disable or enable a gadget. Setting this attribute causes a gadget to become disabled, it gets a ghost pattern and doesn't respond to user input any longer.

Disabled gadgets cannot be activated with the TAB key.

Using `Area.Disabled` on a group of objects will disable all objects within that group.

TYPE
Boolean

APPLICABILITY
ISG

7.12 Area.FixHeight

NAME
Area.FixHeight – fix size of object

FUNCTION
Give your object a fixed pixel height. This tag is absolutely not needed in a general MUI application and only present for emergency situations. Please think twice before using it!

TYPE
Number

APPLICABILITY
I

7.13 Area.FixHeightTxt

NAME
Area.FixHeightTxt – fix size of object

FUNCTION
Give your object a fixed pixel height. The height will match the height of the given string. This tag is absolutely not needed in a general MUI application and only present for emergency situations. Please think twice before using it!

TYPE

String

APPLICABILITY

I

7.14 Area.FixWidth

NAME

Area.FixWidth – fix size of object

FUNCTION

Give your object a fixed pixel width. This tag is absolutely not needed in a general MUI application and only present for emergency situations. Please think twice before using it!

TYPE

Number

APPLICABILITY

I

7.15 Area.FixWidthTxt

NAME

Area.FixWidthTxt – fix size of object

FUNCTION

Give your object a fixed pixel width. The width will match the width of the given string. This tag is absolutely not needed in a general MUI application and only present for emergency situations. Please think twice before using it!

TYPE

String

APPLICABILITY

I

7.16 Area.Font

NAME

Area.Font – adjust object font

FUNCTION

Every MUI object can have its own font, just set it with this tag. Objects without an explicit font setting will inherit it from their parent group.

You normally won't need to open a font yourself, just use one of the predefined values to get a font from the users preferences.

The following fonts are possible:

- Inherit
- Normal
- List
- Tiny
- Fixed
- Title
- Big
- Button

TYPE

String (see above for possible values)

APPLICABILITY

IG

7.17 Area.Frame

NAME

Area.Frame – define object frame

FUNCTION

Define a frame for the current object. Since area class is a superclass for all elements in a window, you can assign frames to every object you wish.

You don't adjust the style of your frame directly, instead you only specify a type:

`None`	For no frame.
`Button`	For standard buttons with text in it.
`ImageButton`	For small buttons with images, e.g. the arrows of a scrollbar.
`Text`	For a text field, e.g. a status line display.
`String`	For a string gadget.
`ReadList`	For a read only list.
`InputList`	For a list that handles input (has a cursor).
`Prop`	For proportional gadgets.
`Gauge`	For gauge gadgets.
`Group`	For groups.
`PopUp`	For popup gadgets.
`Virtual`	For virtual groups.
`Slider`	For slider gadgets.

How the frame is going to look is adjustable via the preferences program.

Four spacing values belong to each frame that tell MUI how many pixels should be left free between the frame and its contents. These spacing values are also user adjustable as long as you don't override them with one of `Area.InnerLeft`, `Area.InnerRight`, `Area.InnerTop`, or `Area.InnerBottom`.

Note: The first object in a window may *not* have a frame. If you need this you will have to create a dummy group with just one child.

TYPE

String (see above for possible values)

APPLICABILITY

I

7.18 Area.FramePhantomHoriz

NAME

Area.FramePhantomHoriz – make a phantom frame

FUNCTION

Setting this to `True` causes the specified frame to be a horizontal phantom frame. The frame will not appear but its vertical components (frame height, inner top and inner bottom spacing) will be used to calculate positions and dimensions (horizontal components are treated as 0).

This is extremely useful for a correct labeling of objects. You would e.g. label a string gadget by using a text object with a phantom string frame. Thus, the label text will be always on the same vertical position as the string gadget text, no matter what spacing values the user configured.

TYPE

Boolean

APPLICABILITY

I

7.19 Area.FrameTitle

NAME

Area.FrameTitle – set frame title

FUNCTION

This tag identifies a text string that will be displayed centered in the top line of a frame. This can become handy if you want to name groups of objects.

You may not use `Area.FrameTitle` without defining a `Area.Frame`.

TYPE

String

APPLICABILITY
I

7.20 Area.Height

NAME
Area.Height – get height of object

FUNCTION
You can use this to read the current position and dimension of an object, if you e.g. need it to pop up some requester below.

Of course, this attribute is only valid when the parent window of the object is currently open.

TYPE
Number

APPLICABILITY
G

7.21 Area.Hide

NAME
Area.Hide – show/hide object

FUNCTION
Objects with this attribute set are not displayed. You can set `Area.Hide` at any time, causing objects to appear and to disappear immediately. A new layout is calculated whenever some objects are shown or hidden. When necessary, MUI will resize the parent window to make place for the new objects.

Currently, MUI does a complete window refresh after showing/hiding objects. This behaviour might get improved in the future.

TYPE
Boolean

APPLICABILITY
ISG

7.22 Area.HorizWeight

NAME
Area.HorizWeight – set/get horizontal object weight

FUNCTION
Adjust the horizontal weight of an object. Usually you can simply use `Area.Weight` instead of this tag but in some two-dimensional groups it may become handy to have different horizontal and vertical weights.

TYPE
Number

APPLICABILITY
ISG

7.23 Area.InnerBottom

NAME
Area.InnerBottom – adjust space between object and frame

FUNCTION
Adjust the space between an object and its frame. Usually you shouldn't use this tag since you will override the users preferred default setting.

TYPE
Number

APPLICABILITY
IG

7.24 Area.InnerLeft

NAME
Area.InnerLeft – adjust space between object and frame

FUNCTION
Adjust the space between an object and its frame. Usually you shouldn't use this tag since you will override the users preferred default setting.

TYPE
Number

APPLICABILITY
IG

7.25 Area.InnerRight

NAME
Area.InnerRight – adjust space between object and frame

FUNCTION
Adjust the space between an object and its frame. Usually you shouldn't use this tag since you will override the users preferred default setting.

TYPE
Number

APPLICABILITY
IG

7.26 Area.InnerTop

NAME

Area.InnerTop – adjust space between object and frame

FUNCTION

Adjust the space between an object and its frame. Usually you shouldn't use this tag since you will override the users preferred default setting.

TYPE

Number

APPLICABILITY

IG

7.27 Area.InputMode

NAME

Area.InputMode – adjust input mode of object (V1.5)

FUNCTION

Adjust the input mode for an object.

MUI has no distinct button class. Instead you can make every object (even groups) behave like a button by setting an input mode for them. Several input modes area available:

`None:` No input, this is not a gadget.

`RelVerify:`
For buttons and similar stuff.

`Immediate:`
Used e.g. in a radio button object.

`Toggle:` For things like checkmark gadgets.

The input mode setting determines how a user action will trigger the attributes `Area.Selected` and `Area.Pressed`. See their documentation for details.

TYPE

String (see above for possible values)

APPLICABILITY

I

7.28 Area.LeftEdge

NAME

Area.LeftEdge – get left edge of object

FUNCTION

You can use this to read the current position and dimension of an object, if you e.g. need it to pop up some requester below.

Of course, this attribute is only valid when the parent window of the object is currently open.

TYPE

Number

APPLICABILITY

G

7.29 Area.MaxHeight

NAME

Area.MaxHeight – set maximum height of object

FUNCTION

Specify a maximum height for an object (in pixels).

TYPE

Number

APPLICABILITY

I

7.30 Area.MaxWidth

NAME

Area.MaxWidth – set maximum width of object

FUNCTION

Specify a maximum width for an object (in pixels).

TYPE

Number

APPLICABILITY

I

7.31 Area.Pressed

NAME

Area.Pressed – learn if button is pressed (V1.5)

FUNCTION

Learn if a button is pressed (or released). The `Area.Pressed` attribute of a gadget is triggered by some user action, depending on the input mode:

`RelVerify:`
: Set when lmb is pressed. Cleared when lmb is released and the mouse is still over the gadget (otherwise it will be cleared too, but without triggering a notification event).

`Immediate:`
: Undefined, use `Area.Selected` for this.

`Toggle:` Undefined, use `Area.Selected` for this.

Waiting for `Area.Pressed` getting `False` is the usual way to react on button gadgets.

TYPE

Boolean

APPLICABILITY

N

7.32 Area.RightEdge

NAME

Area.RightEdge – get right edge of object

FUNCTION

You can use this to read the current position and dimension of an object, if you e.g. need it to pop up some requester below.

Of course, this attribute is only valid when the parent window of the object is currently open.

TYPE

Number

APPLICABILITY

G

7.33 Area.Selected

NAME

Area.Selected – set/get selected state of a gadget (V1.5)

FUNCTION

Get and set the selected state of a gadget. This attribute can be triggered by the user clicking on the gadget (or using the keyboard), depending on the input mode:

`RelVerify:`
: Set when lmb is pressed. Cleared when lmb is released. Cleared when the gadget is selected and the mouse leaves the gadget box. Set when the mouse reenters the gadget box.

`Immediate:`
: Set when lmb is pressed.

`Toggle:` Toggled when lmb is pressed.

Of course you may set this attribute yourself, e.g. to adjust the state of a checkmark gadget.

A selected gadget will display its border reverse and get the configured `SelectedBack` background. This can be avoided using the `Area.ShowSelState` tag.

TYPE

Boolean

APPLICABILITY

ISGN

7.34 Area.ShortHelp

NAME

Area.ShortHelp – set/get bubble help string

FUNCTION

Specify a string that is to be used as bubble help for this object.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

ISG

7.35 Area.ShowSelState

NAME

Area.ShowSelState – configure background mode (V1.5)

FUNCTION

Normally a gadget will reverse its frame and display the configured `SelectedBack` background pattern in its selected state. For some objects (e.g. checkmarks) this is not recommended and can be supressed by setting `Area.ShowSelState` to `False`.

TYPE

Boolean

APPLICABILITY

I

7.36 Area.TopEdge

NAME

Area.TopEdge – get top edge of object

FUNCTION

You can use this to read the current position and dimension of an object, if you e.g. need it to pop up some requester below.

Of course, this attribute is only valid when the parent window of the object is currently open.

TYPE

Number

APPLICABILITY

G

7.37 Area.VertWeight

NAME

Area.VertWeight – set/get vertical object weight

FUNCTION

Adjust the vertical weight of an object. Usually you can simply use `Area.Weight` instead of this tag but in some two-dimensional groups it may become handy to have different horizontal and vertical weights.

TYPE

Number

APPLICABILITY

ISG

7.38 Area.Weight

NAME

Area.Weight – set object weight

FUNCTION

This tag is a shorthand for `Area.HorizWeight` and `Area.VertWeight`, it sets both weights at once.

The weight of an object determines how much room it will get during the layout process. Imagine you have a 100 pixel wide horizontal group with two string gadgets. Usually, each gadget will get half of the room and be 50 pixels wide. If you feel the left gadget is more important and should be bigger, you can give it a weight of 200 (and 100 for the right gadget). Because the left gadget is twice as "heavy" as the right gadget, it will become twice as big (about 66 pixel) as the right one (34 pixel).

Of course giving weights only makes sense if the object is resizable. A `Area.VertWeight` for a (always fixed height) string gadget is useless.

An object with a weight of 0 will always stay at its minimum size.

By default, all objects have a weight of 100.

TYPE

Number

APPLICABILITY

I

7.39 Area.Width

NAME

Area.Width – get width of object

FUNCTION

You can use this to read the current position and dimension of an object, if you e.g. need it to pop up some requester below.

Of course, this attribute is only valid when the parent window of the object is currently open.

TYPE

Number

APPLICABILITY

G

8 Busy class

8.1 Overview

Busy class creates an indefinite progress indicator gadget. This is useful if you need to indicate a busy state without knowing when processing will be finished. For a linear progress indicator, take a look at Gauge class instead. See Section 16.1 [Gauge class], page 97, for details.

This class requires at least MUI Royale 1.4.

8.2 Busy.Move

NAME

Busy.Move – move the busy object (V1.4)

SYNOPSIS

```
mui.DoMethod(id, "Move")
```

FUNCTION

Moves the busy object.

INPUTS

`id`	id of the busy object

8.3 Busy.Speed

NAME

Busy.Speed – set/get animation speed (V1.4)

FUNCTION

Setting this attribute will trigger the speed of the moving busy bar. Valid value range is 1 to 250 (milliseconds).

Additionally, there are the following special values:

`Off`	Stop the busy bar.
`User`	Use the user defined speed from the preferences.

TYPE

Number or string (see above for possible values)

APPLICABILITY

ISG

9 Button class

9.1 Overview

Button class allows you to create buttons very easily. If the button label contains an underscore, MUI Royale will automatically set up the character following this underscore as a keyboard shortcut. If you don't want this behaviour, set `Button.NoAutoKey` to `True`.

Example:

```
<button id="ok" notify="pressed">OK</button>
```

The button name you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

9.2 Button.HiChar

NAME

Button.HiChar – highlight a character in button label

FUNCTION

If the character given here exists in the displayed string (no matter if upper or lower case), it will be underlined. Highlighting a character in the button label is useful in connection with `Area.ControlChar`.

TYPE

Single character string

APPLICABILITY

I

9.3 Button.Label

NAME

Button.Label – set/get button label

FUNCTION

Set or get the button's label using this attribute.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

SG

9.4 Button.NoAutoKey

NAME

Button.NoAutoKey – disable auto button shortcut

FUNCTION

If the button label contains an underscore, MUI Royale will automatically set up the character following this underscore as a keyboard shortcut by default. I.e. if the button label is `"_OK"`, then `"o"` will automatically become the keyboard shortcut for this button. If you don't want this behaviour, set this attribute to `True`.

Defaults to `False`.

TYPE

Boolean

APPLICABILITY

I

9.5 Button.Pressed

NAME

Button.Pressed – learn if a button is pressed

FUNCTION

This attribute is triggered if the user presses the button.

TYPE

Boolean

APPLICABILITY

N

9.6 Button.Selected

NAME

Button.Selected – toggle selection state

FUNCTION

Use this to toggle the selection state of a toggle button or get notified of a user toggle event. If you want to use this attribute, you have to set `Button.Toggle` to `True` first.

TYPE

Boolean

APPLICABILITY

ISGN

9.7 Button.Toggle

NAME

Button.Toggle – create toggle button

FUNCTION

If you set this to `True`, the button will be a toggle button that can have two states: On and off. You can use the `Button.Selected` attribute to toggle states and get notified about user toggle events.

By default, the button will be a normal button.

TYPE

Boolean

APPLICABILITY

I

10 Checkmark class

10.1 Overview

Checkmark class creates an image button with a checkmark that can have two different states. It is often used to toggle options. This class really just creates the checkmark button. It does not put a label next to it. If you want to have a label for your checkmark, you need to do this manually by placing a text or label object next to the checkmark.

Here is an example of how to create a checkmark with an adjacent label:

```
<hgroup>
   <label>Enable cool option</label>
   <checkmark/>
</hgroup>
```

10.2 Checkmark.Selected

NAME

Checkmark.Selected – set/get checkmark state

FUNCTION

Get and set the selected state of a checkmark. This attribute can be triggered by the user clicking on the gadget or using the keyboard. Of course you may set this attribute yourself, e.g. to adjust the state of a checkmark gadget.

You can also set up a notification on this attribute to learn when the user toggles the checkmark state.

TYPE

Boolean

APPLICABILITY

ISGN

11 Coloradjust class

11.1 Overview

Coloradjust class creates some gadgets that allow adjusting a single color. Depending on the operating system, different kinds of gadgets are be used. Kickstart 2.x users might only receive an RGB slider triple, Kickstart 3.x users could get an additional colorwheel if available. However, the outfit of this class is not important for you as a programmer.

Here is an example XML excerpt for creating a coloradjust gadget set to all white:

```
<coloradjust rgb="#ffffff"/>
```

11.2 Coloradjust.RGB

NAME

Coloradjust.RGB – set/get color

FUNCTION

Set or get the adjusted color. If you set up a notification on this attribute, you will be notified whenever the color changes. From the Hollywood script the color is specified as a simple numerical value containing 8 bits for each component. When you specify the color in the XML file, it has to be passed as a 6 character string prefixed by the #-character (just like in HTML).

TYPE

Number

APPLICABILITY

ISGN

12 Colorfield class

12.1 Overview

Colorfield class creates a rectangle filled with a specific color, useful e.g. within a palette requester. You can change the color of the field at any time by setting its RGB attributes.

The field will try to obtain an exclusive pen on the current screen. When none is available, it just displays some kind of rastered background. Maybe it will get a little more intelligent and try to display the color by mixing together some other colors, but that's a future topic.

Needless to say that Colorfield only works with Kickstart 3.x and above, since lower operating systems don't support pen sharing. When using this class with a lower OS, you will also get some kind of (boring) raster.

Here is an example XML excerpt for creating a colorfield gadget set to all white:

```
<colorfield rgb="#ffffff"/>
```

12.2 Colorfield.RGB

NAME

Colorfield.RGB – set/get color

FUNCTION

Set or get the current color. If you set up a notification on this attribute, you will be notified whenever the color changes. From the Hollywood script the color is specified as a simple numerical value containing 8 bits for each component. When you specify the color in the XML file, it has to be passed as a 6 character string prefixed by the #-character (just like in HTML).

TYPE

Number

APPLICABILITY

ISGN

13 Cycle class

13.1 Overview

Cycle class generates the well known cycle gadgets. However, MUI cycle gadgets feature a (configurable) popup menu to avoid clicking through many entries.

When you declare a cycle gadget, you have to use the `<item>` tag to fill the cycle gadget with items. Every cycle gadget needs to have at least one item.

Here is an example XML excerpt for creating a cycle gadget:

```
<cycle id="printer">
    <item>HP Deskjet</item>
    <item>NEC P6</item>
    <item>Okimate 20</item>
</cycle>
```

The cycle entries that you specify using the `<item>` tag can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

13.2 Cycle.Active

NAME

Cycle.Active – set/get active cycle item

FUNCTION

This attributes defines the number of the active entry in the cycle gadgets. Valid range is from 0 for the first entry to NumEntries-1 for the last.

Setting `Cycle.Active` causes the gadget to be updated. On the other hand, when the user plays around with the gadget, `Cycle.Active` will always reflects the current state.

Passing the special values `Next` or `Prev` during set causes the gadget to cycle through its entries in the given direction.

TYPE

Number or string (see above for possible values)

APPLICABILITY

ISGN

14 Dirlist class

14.1 Overview

Dirlist class provides a quick and easy way of showing entries in a directory. It features lots of control attributes, many of them known from the popular asl file requester.

This class is *not* intended to replace asl.library! Nobody wants to see every MUI application coming with another selfmade file requester. Please continue using ASL for real file requesting purposes!

However, sometimes it may be useful to have a little directory list placed somewhere in your user interface. Imagine an answering machine tool that stores incoming calls in a preconfigured directory. Using a dirlist object, you can include the GUI for selecting a call in your window with lots of other gadgets like "Play", "Delete", etc.

Dirlist class creates a listview with six columns representing all files attributes: Name, size, date, time, flags and comment. If you do not want to have all of these attributes displayed, you can use Listviewcolumn class to make adjustments.

Dirlist class is a subclass of listview class. Thus, you can use most attributes and methods of listview class on it too. For example, if you want to read the entries of your directory, just send the dirlist object a `Listview.GetEntry` method, or if you want to listen to changes in the active list entry, set up a notification on `Listview.Active`.

If you use `Listview.GetEntry` on this class, you will get six return values representing the six different columns of the dirlist listview. Note that you will always get six return values even if you made adjustments to the column display using Listviewcolumn class. The protection flags are returned as a single number containing a bitfield combination of the `#FILEATTR_XXX` constants from Hollywood.

When creating a dirlist object in XML code, you always have to add at least one column to it. This is done by using the Listviewcolumn class. Here is an example of a minimal dirlist declaration with just a single column (the name column, that is):

```
<dirlist>
   <column/>  <!-- Name -->
</dirlist>
```

Here is a declaration that includes all six columns:

```
<dirlist>
   <column/>  <!-- Name -->
   <column/>  <!-- Size -->
   <column/>  <!-- Date -->
   <column/>  <!-- Time -->
   <column/>  <!-- Flags -->
   <column/>  <!-- Comment -->
</dirlist>
```

If you only want to have the name column and the comment column displayed, you can use the `Listviewcolumn.Col` attribute to achieve this:

```
<dirlist>
   <column/>          <!-- Name -->
```

```
        <column col="5"/>  <!-- Comment -->
     </dirlist>
```

To reverse the order of columns use:

```
     <dirlist>
        <column col="5"/>  <!-- Comment -->
        <column col="4"/>  <!-- Flags -->
        <column col="3"/>  <!-- Time -->
        <column col="2"/>  <!-- Date -->
        <column col="1"/>  <!-- Size -->
        <column col="0"/>  <!-- Name -->
     </dirlist>
```

See Section 25.1 [Listview class], page 143, for details.

See Section 26.1 [Listviewcolumn class], page 159, for details.

14.2 Dirlist.AcceptPattern

NAME

Dirlist.AcceptPattern – set filter pattern

FUNCTION

Set a filter pattern specified in the AmigaDOS pattern format. Entries not matching this pattern are rejected.

TYPE

String

APPLICABILITY

IS

14.3 Dirlist.Directory

NAME

Dirlist.Directory – set/get active directory

FUNCTION

Set a new directory for the dirlist object. Since reading a directory can take a long long time, MUI delegates this work to a sub task.

Setting this attribute causes the object to clear the current directory (if any) and start loading a new one. `Dirlist.Status` will be set to `Reading` and the sub task will be launched.

By listening to `Dirlist.Status`, you can learn if the directory reading is completed or if something went wrong.

Passing the empty string here (`""`) just clears the current directory and sets `Dirlist.Status` to `Invalid`.

Defaults to `SYS:`.

TYPE

String

APPLICABILITY
ISG

14.4 Dirlist.DrawersOnly

NAME
Dirlist.DrawersOnly – indicate whether you want only drawers displayed

FUNCTION
Indicate whether you only want drawers to be displayed.

TYPE
Boolean

APPLICABILITY
IS

14.5 Dirlist.FilesOnly

NAME
Dirlist.FilesOnly – indicate whether you want only files displayed

FUNCTION
Indicate whether you only want files to be displayed.

TYPE
Boolean

APPLICABILITY
IS

14.6 Dirlist.FilterDrawers

NAME
Dirlist.FilterDrawers – apply pattern matching to drawers too

FUNCTION
Indicate whether you want drawers matched agains `Dirlist.RejectPattern` and `Dirlist.AcceptPattern`.

Defaults to `False`.

TYPE
Boolean

APPLICABILITY
IS

14.7 Dirlist.MultiSelDirs

NAME

Dirlist.MultiSelDirs – allow multiple selection of directories

FUNCTION

Allows multi selection of directories.

Defaults to `False`.

TYPE

Boolean

APPLICABILITY

IS

14.8 Dirlist.NumBytes

NAME

Dirlist.NumBytes – get number of bytes in directory

FUNCTION

When `Dirlist.Status` is `Valid`, you can obtain the number of bytes occupied by the directory from this tag.

TYPE

Number

APPLICABILITY

G

14.9 Dirlist.NumDrawers

NAME

Dirlist.NumDrawers – get number of drawers in directory

FUNCTION

When `Dirlist.Status` is `Valid`, you can obtain the number of drawers in the displayed directory from this tag.

TYPE

Number

APPLICABILITY

G

14.10 Dirlist.NumFiles

NAME

Dirlist.NumFiles – get number of files in directory

FUNCTION

When `Dirlist.Status` is `Valid`, you can obtain the number of files in the displayed directory from this tag.

TYPE

Number

APPLICABILITY

G

14.11 Dirlist.Path

NAME

Dirlist.Path – get full path of active entry

FUNCTION

When `Dirlist.Status` is `Valid` and you have an active entry in the list (`Listview.Active` not equal `Off`), you will receive a pointer to the complete path specification of the selected file. Otherwise you get an empty string.

TYPE

String

APPLICABILITY

G

14.12 Dirlist.RejectIcons

NAME

Dirlist.RejectIcons – indicate whether to reject icons

FUNCTION

Indicate whether you want icons (*.info files) to be rejected.

TYPE

Boolean

APPLICABILITY

IS

14.13 Dirlist.RejectPattern

NAME

Dirlist.RejectPattern – set filter pattern

FUNCTION

Set a filter pattern specified in the AmigaDOS pattern format. Entries matching this pattern are rejected.

TYPE

String

APPLICABILITY

IS

14.14 Dirlist.ReRead

NAME

Dirlist.ReRead – refresh dirlist object

SYNOPSIS

```
mui.DoMethod(id, "ReRead")
```

FUNCTION

Force the dirlist object to reread the current directory.

INPUTS

`id`	id of the dirlist object

14.15 Dirlist.SortDirs

NAME

Dirlist.SortDirs – set where directories shall be displayed

FUNCTION

Adjust the place where directories shall be displayed.

The following values are possible:

`First`	Display directories at the top.
`Last`	Display directories at the bottom.
`Mix`	Interleave directory and file display.

TYPE

String (see above for possible values)

APPLICABILITY

IS

14.16 Dirlist.SortHighLow

NAME

Dirlist.SortHighLow – reverse sorting mode

FUNCTION

Indicate if you want to sort your directory reversely.

TYPE

Boolean

APPLICABILITY

IS

14.17 Dirlist.SortType

NAME

Dirlist.SortType – indicate sorting criteria

FUNCTION

Indicate what fields should be used as sort criteria.

The following values are possible:

`Name`	Sort by name.
`Date`	Sort by date.
`Size`	Sort by size.

TYPE

String (see above for possible values)

APPLICABILITY

IS

14.18 Dirlist.Status

NAME

Dirlist.Status – get status of dirlist object

FUNCTION

Read the status of the dirlist object. The result is one of

`Invalid`	Object contains no valid directory.
`Reading`	Object is currently reading a new directory.
`Valid`	Object contains a valid directory.

TYPE

String (see above for possible values)

APPLICABILITY

GN

14.19 Dirlist.Title

NAME

Dirlist.Title – show/hide list title

FUNCTION

Specify whether you want to have title bar for the listview. The title is displayed at the very first line and doesn't scroll away when the list top position moves.

TYPE

Boolean

APPLICABILITY

ISG

15 Floattext class

15.1 Overview

Floattext class is a class that takes a big text string as input and splits it up into several lines to be dislayed. Formatting capabilities include paragraphs and justified text with word wrap.

Here is an example of how to use the `<floattext>` command:

```
<floattext>Hello World</floattext>
```

Here is the same example in bold:

```
<floattext>\33bHello World</floattext>
```

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

15.2 Floattext.Justify

NAME

Floattext.Justify – set text alignment

FUNCTION

Indicate whether you want your the text aligned to the left and right border. MUI will try to insert spaces between words to reach this goal.

TYPE

Boolean

APPLICABILITY

ISG

15.3 Floattext.TabSize

NAME

Floattext.TabSize – adjust tab size

FUNCTION

Adjust the tab size for a text. The tab size is measured in spaces, so if you plan to use tabs not only at the beginning of a paragraph, you should consider using the fixed width font.

Tab size defaults to 8.

TYPE

Number

APPLICABILITY

IS

15.4 Floattext.Text

NAME

Floattext.Text – set/get contents of floattext object

FUNCTION

String of characters to be displayed as floattext. This string may contain linefeeds to mark the end of paragraphs or tab characters for indention.

MUI will automatically format the text according to the width of the floattext object. If a word won't fit into the current line, it will be wrapped.

If you plan to use tabs not only at the beginning of a line you should consider using the configured fixed width font.

Please note that justification and word wrap with proportional fonts is a complicated operation and may take a considerable amount of time, especially with long texts on slow machines.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

SG

16 Gauge class

16.1 Overview

A gauge object is a nice looking display element useful for some kind of progress display. It is often used together with Scale class to create nice progress bars.

Here is an example XML declaration:

```
<vgroup>
   <gauge horiz="true" infotext="...%ld %% completed..."/>
   <scale/>
</vgroup>
```

16.2 Gauge.Current

NAME
Gauge.Current – set/get current level

FUNCTION
Set the current level of the gauge. The value must be between 0 and `Gauge.Max`.

TYPE
Number

APPLICABILITY
ISGN

16.3 Gauge.Divide

NAME
Gauge.Divide – set divisor

FUNCTION
If this attribute is not 0, every value set with `Gauge.Current` will be divided by this before further processing.

TYPE
Number

APPLICABILITY
ISG

16.4 Gauge.Horiz

NAME
Gauge.Horiz – set gauge alignment

FUNCTION
Determine if you want a horizontal or vertical gauge. Default to `False`.

TYPE

Boolean

APPLICABILITY

I

16.5 Gauge.InfoText

NAME

Gauge.InfoText – set/get gauge info text

FUNCTION

The text given here is displayed within a gauge object and is usually intended to show some kind of percentage information.

This texts preparse is set to `"\33c\0338"`, this makes it appear centered and highlighted by default.

Any %ld will be replaced with the current value of `Gauge.Current`.

Note that if you want to have a single percent character (%) you need to escape it using another percent character (%%) because the single percent character is used as a wildcard character.

Note: Up to V18 of gauge.mui, InfoText worked only for horizontal gauges. If you intend to use and change info text, you should specify an empty `Gauge.InfoText` (`""`) at object creation time. This makes your object get a fixed height that fits the height of the info text.

Since version 19 of gauge.mui, you can also use `Gauge.InfoText` for vertical gauges.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

ISG

EXAMPLE

```
mui.Set(obj, "InfoText", "Please wait... (%ld %%)")
```

The code above sets up a custom gauge text consisting of the line `"Please wait"` together with a percentage specification indicating the current gauge level.

16.6 Gauge.Max

NAME

Gauge.Max – set maximum value of gauge

FUNCTION

Set the maximum value for the gauge. Defaults to 100.

Currently, `Gauge.Max` and `Gauge.Current` is limited to 16 bit.

TYPE
Number

APPLICABILITY
ISG

17 Group class

17.1 Overview

Group class is responsible for the complete layout of a MUI window. A group may contain any number of child objects, maybe buttons, cycle gadgets or even other groups.

Some attributes of group class define how the children of a group are layouted. You can e.g. tell your group to place its children horizontally (in a row) or vertically (in a column). Since every MUI object knows about its minimum and maximum dimensions, group class has everything it needs to do that job.

More sophisticated layout is possible by assigning different weights to objects in a group or by making a group two-dimensional.

Beneath the layout issues, a group object passes attributes and methods through to all of its children. Thus, you can talk and listen to any child of a group by talking and listening to the group itself.

The following different group types are supported by MUI Royale:

`<hgroup>` Group children will be laid out in a row (vertical).

`<vgroup>` Group children will be laid out in a column (horizontal).

`<colgroup>`
Group children will be laid out in columns.

`<virtgroup>`
A virtual group. Normally used inside a scrollgroup. See Section 50.1 [Virtgroup class], page 277, for details.

`<scrollgroup>`
A group with scrollbars. Virtual groups are usually embedded inside a scrollgroup. See Section 43.1 [Scrollgroup class], page 217, for details.

Column groups are useful if you need to have identical gadget sizes for all your children in a group for a more pleasant visual appearance. For example, imagine a form made up of string gadgets and text objects. It is recommended to use a `<colgroup>` here because it leads to a clear and ordered visual appearance. Here's an example:

```
<colgroup columns="2">
   <text>Name</text>
   <string/>
   <text>Street</text>
   <string/>
   <text>City</text>
   <string/>
   <text>Zip code</text>
   <string/>
   <text>Country</text>
   <string/>
   <text>Telephone</text>
   <string/>
```

```
        <text>Email</text>
        <string/>
     </colgroup>
```

If we used a `<vgroup>` with one `<hgroup>` per row the appearance would be pretty bad because the string gadget's width would be different for each line which looks pretty non-professional.

17.2 Group.ActivePage

NAME

Group.ActivePage – set/get active page of page group

FUNCTION

Set (or get) the active page of a page group. Only this active page is displayed, all others are hidden.

The value may range from 0 (for the first child) to numchildren-1 (for the last child). Children are adressed in the order of creation.

The following special values are possible when setting the active page:

`First`	First page.
`Last`	Last page.
`Prev`	Previous page.
`Next`	Next page.
`Advance`	Advance page.

Note: You may *never* supply an incorrect page value!

This attribute can also be used in connection with Register class. See Section 40.1 [Register class], page 207, for details.

TYPE

Number or string (see above for possible values)

APPLICABILITY

ISGN

17.3 Group.AddHead

NAME

Group.AddHead – add detached object as first group child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddHead", obj)
```

FUNCTION

This method can be used to add the detached object specified by "obj" to the group object specified by "id". The detached object will be added as the group's first child. After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Before you can call this method, you have to put the group into a special state that allows the addition and subtraction of children. This can be done by running the `Group.InitChange` and `Group.ExitChange` methods on the respective group object.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Group.Remove` method.

INPUTS

`id`	id of the group object
`obj`	id of the object to attach

EXAMPLE

See Section 5.2 [mui.CreateObject], page 36.

17.4 Group.AddTail

NAME

Group.AddTail – add detached object as last group child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddTail", obj)
```

FUNCTION

This method can be used to add the detached object specified by "obj" to the group object specified by "id". The detached object will be added as the group's last child. After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Before you can call this method, you have to put the group into a special state that allows the addition and subtraction of children. This can be done by running the `Group.InitChange` and `Group.ExitChange` methods on the respective group object.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Group.Remove` method.

INPUTS

`id`	id of the group object
`obj`	id of the object to attach

EXAMPLE

See Section 5.2 [mui.CreateObject], page 36.

17.5 Group.ChildList

NAME

Group.ChildList – get a list of all group children (V1.2)

FUNCTION

This attribute allows you to get a table that contains information about all children that are currently attached the group. For each child in the group, the following table elements will be initialized:

`ID:`	Contains the ID of the group child.
`Class:`	Contains the name of the class this object belongs to, e.g. "Cycle".
`MUIClass:`	Contains the internal MUI class name for this object, e.g. "Cycle.mui". This is probably of not much interest for your application.

TYPE

Table

APPLICABILITY

G

EXAMPLE

```
t = mui.Get("mygroup", "childlist")
For Local k = 0 To ListItems(t) - 1
   DebugPrint(t[k].id, t[k].class, t[k].muiclass)
Next
```

The code above dumps all objects that belong to "mygroup".

17.6 Group.Columns

NAME

Group.Columns – define group columns

FUNCTION

Indicate number of columns in a two dimensional group. If you use this tag, the total number of children must be dividable by the number of columns.

The children will be positioned in a two dimensional array, e.g. allowing easy creation of button fields (maybe for calculator).

The children of your group are always read line by line.

When MUI layouts two-dimensional groups, it does actually two layout calculations, one for the rows and one the columns. Parameters like weights and dimensions are handled this way:

- the minimum width of a column/row is the maximum minimum width of all objects in this column/row.
- the maximum width of a column/row is the minimum maximum width of all objects in this column/row.
- the weight of a column/row is the sum of all objects in this column/row.

TYPE

Number

APPLICABILITY

I

17.7 Group.ExitChange

NAME

Group.ExitChange – terminate group exchange state (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "ExitChange", force)
```

FUNCTION

This method terminates the state established by `Group.InitChange`. If children have been added or removed, MUI will refresh the group making the changes visible to the user. You can force MUI to do this refresh by setting the "force" argument to `True`. In that case MUI will always refresh the whole group no matter if objects have been added or removed. Forcing a refresh is useful if there's an object inside your group that you want to force a refresh on.

INPUTS

`id`	id of the group object
`force`	specify `True` here to force a complete refresh; otherwise MUI will only refresh the group if objects have been added or removed

EXAMPLE

See Section 5.2 [mui.CreateObject], page 36.

17.8 Group.HorizSpacing

NAME

Group.HorizSpacing – set/get horizontal spacing

FUNCTION

Number of pixels to be inserted between horizontal elements of a group.

Please use this tag wisely, you will override the user's prefered default setting!

TYPE

Number

APPLICABILITY

ISG

17.9 Group.InitChange

NAME

Group.InitChange – prepare group for addition or removal of children (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "InitChange")
```

FUNCTION

Prepares a group for dynamic adding/removing of objects. Since version 3 MUI offers the possibility to dynamically add/remove children from groups, even when the window that contains these objects is currently open. To be able to do this, you must first put the group into a special "exchange" state by using this method. Then, you can add/remove children at will. If you're done, use `Group.ExitChange` to make MUI recalculate the display.

INPUTS

`id`	id of the group object

EXAMPLE

See Section 5.2 [mui.CreateObject], page 36.

17.10 Group.Insert

NAME

Group.Insert – insert detached object after specified child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "Insert", obj, pred)
```

FUNCTION

This method can be used to insert the detached object specified by "obj" to the group object specified by "id". The detached object will be added after the child specified by "pred". After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Before you can call this method, you have to put the group into a special state that allows the addition and subtraction of children. This can be done by running the `Group.InitChange` and `Group.ExitChange` methods on the respective group object.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Group.Remove` method.

INPUTS

`id`	id of the group object
`obj`	id of the object to insert
`pred`	the object will be inserted after this object

EXAMPLE

See Section 5.2 [mui.CreateObject], page 36.

17.11 Group.PageMode

NAME

Group.PageMode – put group in page mode

FUNCTION

Settings this attribute to `True` makes the current group a page group. Page groups always display only one of their children. Which one can be adjusted with the `Group.ActivePage` attribute.

Imagine you have a preferences window with several different pages, e.g. the MUI preferences with object, frame, image, font, screen, keyboard and system prefs. Instead of one separate window for each group, you could put all pages into a page group and have a cycle gadget for page switching. This will make your program easier to use since the user won't have to handle a lot of windows. However, he will not be able to work with more than one page at the same time.

Sizes are calculated as follows:

The minimum width/height of a page group is the maximum minimum width/height of all its children.

The maximum width/height of a page group is the minimum maximum width/height of all its children.

When the maximum width/height of a child in a page group is smaller than the minimum width/height of the page group (since it contains another child with big minimum width/height), the child be centered.

Page groups are not limited in depth, children of a page group may of course be other page groups.

If you want to have a gadget only visible under certain conditions, you could make a page group containing this gadget and an empty rectangle object.

If you want TAB cycling for the objects in a page group, simply include all objects in the cycle chain (as if they all were visible at the same time).

A special type of page group is created by Register class which generates a series of register tabs for all of its children. See Section 40.1 [Register class], page 207, for details.

TYPE

Boolean

APPLICABILITY

I

17.12 Group.Remove

NAME

Group.Remove – detach object from group (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "Remove", obj)
```

FUNCTION

This method can be used to detach the specified object from the specified group. After this method returns the specified object will change its state from attached to detached. This means that you could now attach it to another group using a function like `Group.Insert` or you could free it using `mui.FreeObject()`.

Before you can call this method, you have to put the group into a special state that allows the addition and subtraction of children. This can be done by running the `Group.InitChange` and `Group.ExitChange` methods on the respective group object.

INPUTS

`id`	id of the group object
`obj`	id of the object to remove

EXAMPLE

```
mui.DoMethod("mygroup", "initchange")
mui.DoMethod("mygroup", "remove", "mychild")
mui.DoMethod("mygroup", "exitchange", false)
```

The code above removes the child "mychild" from the group "mygroup". You could then attach "mychild" to another group or free it.

17.13 Group.SameHeight

NAME

Group.SameHeight – set same height for all children

FUNCTION

Boolean value to indicate that all children of this group shall have the same height.

TYPE

Boolean

APPLICABILITY

I

17.14 Group.SameSize

NAME

Group.SameSize – set same size for all children

FUNCTION

This is a shorthand for `Group.SameWidth` and `Group.SameHeight`, it sets both of these attributes at once.

Using `Group.SameSize`, you won't need to think if your group is horizontal or vertical, both cases are handled automatically.

Forcing all objects of a group to be the same size is e.g. useful for a row of buttons. It's visually more attractive when these buttons have equal sizes instead of being just as big as the text within.

TYPE

Boolean

APPLICABILITY

I

17.15 Group.SameWidth

NAME

Group.SameWidth – set same width for all children

FUNCTION

Boolean value to indicate that all children of this group shall have the same width.

TYPE

Boolean

APPLICABILITY

I

17.16 Group.Spacing

NAME

Group.Spacing – set/get spacing

FUNCTION

This is a shorthand for `Group.HorizSpacing` and `Group.VertSpacing`, it sets both of these attributes at once.

Using `Group.Spacing`, you won't need to think if your group is horizontal or vertical, both cases are handled automatically.

Note that setting a spacing value for a group overrides the user's default settings. Please use it only if you have a good reason.

TYPE

Number

APPLICABILITY

ISG

17.17 Group.Title

NAME

Group.Title – set group title

FUNCTION

If you define groups for a register, you can use this attribute to specify a title for this group that shall be displayed in the register. See Section 40.1 [Register class], page 207, for details.

On MUI 3.x this attribute can only be specified during initialization (applicability is just I on MUI 3.x).

If MUI 4 and MUI Royale 1.2 are available, then you can also use `mui.Set()` and `mui.Get()` with this attribute (ISG applicability).

Furthermore, if MUI 4 and MUI Royale 1.2 are available the string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

ISG

17.18 Group.VertSpacing

NAME

Group.VertSpacing – set/get vertical spacing

FUNCTION

Number of pixels to be inserted between vertical elements of a group.

Please use this tag wisely, you will override the user's prefered default setting!

TYPE

Number

APPLICABILITY

ISG

18 Hollywood class

18.1 Overview

Hollywood class is a powerful MUI class that allows you to embed a complete Hollywood display inside your MUI GUI. Whenever you draw something to a Hollywood display that is attached to Hollywood class, it will automatically be drawn to your MUI GUI as well. You can even hide the Hollywood display and it will still work. Furthermore, all mouse clicks and key strokes that happen inside Hollywood class will be forwarded to the corresponding Hollywood display as normal Hollywood events. Thus, Hollywood class allows you to use almost all of Hollywood's powerful features inside a MUI GUI as well.

Here is an example of how to embed Hollywood display 1 inside your GUI:

```
<hollywood display="1"/>
```

See Section 3.14 [Hollywood bridge], page 18, for details.

18.2 Hollywood.Display

NAME

Hollywood.Display – set Hollywood display to use

FUNCTION

Attach a Hollywood display to the object. You have to specify the identifier of a Hollywood display here.

This attribute is mandatory and has to be specified whenever you create a Hollywood object. You cannot create an empty Hollywood object without an attached display.

TYPE

Number

APPLICABILITY

IS

18.3 Hollywood.MaxHeight

NAME

Hollywood.MaxHeight – set/get maximum height

FUNCTION

If you set this attribute, your Hollywood object will become resizable. Set this attribute to the maximum allowable height. The maximum height that you can specify here is 16384 pixels.

To indicate that your Hollywood object has no maximum size, you should set this attribute to the maximum allowable size of 16384 pixels. This should be sufficiently large enough for most purposes.

Whenever the user changes the size of your Hollywood object, the Hollywood display that is attached to this Hollywood object will get a "SizeWindow" event which you can listen to using the `InstallEventHandler()` function.

TYPE

Number

APPLICABILITY

ISG

18.4 Hollywood.MaxWidth

NAME

Hollywood.MaxWidth – set/get maximum width

FUNCTION

If you set this attribute, your Hollywood object will become resizable. Set this attribute to the maximum allowable width. The maximum width that you can specify here is 16384 pixels.

To indicate that your Hollywood object has no maximum size, you should set this attribute to the maximum allowable size of 16384 pixels. This should be sufficiently large enough for most purposes.

Whenever the user changes the size of your Hollywood object, the Hollywood display that is attached to this Hollywood object will get a "SizeWindow" event which you can listen to using the `InstallEventHandler()` function.

TYPE

Number

APPLICABILITY

ISG

18.5 Hollywood.MinHeight

NAME

Hollywood.MinHeight – set/get minimum height

FUNCTION

If you set this attribute, your Hollywood object will become resizable. Set this attribute to the minimum allowable height.

Whenever the user changes the size of your Hollywood object, the Hollywood display that is attached to this Hollywood object will get a "SizeWindow" event which you can listen to using the `InstallEventHandler()` function.

TYPE

Number

APPLICABILITY

ISG

18.6 Hollywood.MinWidth

NAME

Hollywood.MinWidth – set/get minimum width

FUNCTION

If you set this attribute, your Hollywood object will become resizable. Set this attribute to the minimum allowable width.

Whenever the user changes the size of your Hollywood object, the Hollywood display that is attached to this Hollywood object will get a "SizeWindow" event which you can listen to using the `InstallEventHandler()` function.

TYPE

Number

APPLICABILITY

ISG

19 HSpace class

19.1 Overview

HSpace class is a subclass of rectangle class and simply creates some empty horizontal space between two objects.

19.2 HSpace.Width

NAME
HSpace.Width – set horizontal space

FUNCTION
Sets the desired horizontal space for this object in pixels.

TYPE
Number

APPLICABILITY
I

20 Image class

20.1 Overview

Image class is used to display one of MUI's standard images or some selfmade image data.

Here is an example XML declaration:

```
<image source="brush:1"/>
```

The XML code above creates an image object from Hollywood brush number 1. Image class fully supports mask and alpha channel transparency in Hollywood brushes so you can also embed images with transparent areas.

20.2 Image.Brush

NAME

Image.Brush – change image to new brush

FUNCTION

If `Image.Source` is set to `Brush`, you can use this attribute to change the brush that shall be displayed by the image object. Simply pass the identifier of the new Hollywood brush you would like to have displayed.

Image class fully supports mask and alpha channel transparency in Hollywood brushes so you can also embed images with transparent areas.

TYPE

Number

APPLICABILITY

S

20.3 Image.FreeHoriz

NAME

Image.FreeHoriz – allow horizontal scaling

FUNCTION

Tell the image if its allowed to get scaled horizontally. Defaults to `False`.

TYPE

Boolean

APPLICABILITY

I

20.4 Image.FreeVert

NAME

Image.FreeVert – allow vertical scaling

FUNCTION

Tell the image if its allowed to get scaled vertically. Defaults to `False`.

TYPE

Boolean

APPLICABILITY

I

20.5 Image.Source

NAME

Image.Source – set image to display

FUNCTION

Set the image to display. The following predefined MUI images are available here:

- ArrowDown
- ArrowLeft
- ArrowRight
- ArrowUp
- Assign
- Brush:<id>
- CheckMark
- Chip
- Cycle
- Disk
- Drawer
- HardDisk
- ListCursor
- ListSelCur
- ListSelect
- Network
- PopDrawer
- PopFile
- PopUp
- PropKnob
- RadioButton
- SliderKnob

- TapeDown
- TapePause
- TapePlay
- TapePlayback
- TapeRecord
- TapeStop
- TapeUp
- Volume

If you use the type `Brush` you need to set `<id>` to the identifier of a Hollywood brush that you want to use for this image object.

TYPE

String (see above for possible values)

APPLICABILITY

I

20.6 Image.State

NAME

Image.State – set image state

FUNCTION

Some MUI images offer different states, you can select one of the by setting this attribute. Simply use one of the following values to switch the state:

`Normal`	Normal state.
`Selected`	Selected state.
`Disabled`	Disabled state.
`Busy`	Busy state.
`Indeterminate`	Indeterminate state.
`InactiveNormal`	Normal inactive state.
`InactiveSelected`	Selected inactive state.
`InactiveDisabled`	Disabled inactive state.
`SelectedDisabled`	Selected disabled state.

TYPE

String (see above for possible values)

APPLICABILITY

IS

21 Imagebutton class

21.1 Overview

Imagebutton class is a subclass of image class. It turns an image object into an image button that can be pressed by the user. Imagebutton class recognizes all attributes of image class.

This class is probably not of much use since you can include images in button labels using text formatting codes anyway. That is why you should use the normal button class instead.

See Section 9.1 [Button class], page 75, for details.

See Section 3.9 [Text formatting codes], page 14, for details.

21.2 Imagebutton.Pressed

NAME

Imagebutton.Pressed – learn if an image button is pressed

FUNCTION

This attribute is triggered if the user presses the button.

TYPE

Boolean

APPLICABILITY

N

22 Label class

22.1 Overview

Label class is a superclass of text class and allows you to easily create labels for other MUI objects like checkmarks or cycle objects.

Here is an example of how to use the `<label>` command:

```
<label>Hello World</label>
```

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

Please note that labels are not resizable horizontally. That is why they can easily block resizing of your whole GUI. To circumvent this problem, you can simply put the label in a `<hgroup>` together with an empty `<rectangle>` object. See Section 39.1 [Rectangle class], page 205, for details.

22.2 Label.Centered

NAME

Label.Centered – center label text

FUNCTION

Set this to `True` to center the label text.

TYPE

Boolean

APPLICABILITY

I

22.3 Label.DoubleFrame

NAME

Label.DoubleFrame – use double high frame for label

FUNCTION

Set this to `True` to have this label use a double high frame.

TYPE

Boolean

APPLICABILITY

I

22.4 Label.FreeVert

NAME
Label.FreeVert – allow vertical resizing

FUNCTION
Set this to `True` to allow vertical resizing of this label. Defaults to `False`.

TYPE
Boolean

APPLICABILITY
I

22.5 Label.Key

NAME
Label.Key – highlight a character in label

FUNCTION
If the character given here exists in the displayed string (no matter if upper or lower case), it will be underlined. Highlighting a character in the label is useful in connection with `Area.ControlChar`.

TYPE
Single character string

APPLICABILITY
I

22.6 Label.LeftAligned

NAME
Label.LeftAligned – set text alignment

FUNCTION
Set this to `True` to have this label use left aligned text.

TYPE
Boolean

APPLICABILITY
I

22.7 Label.SingleFrame

NAME
Label.SingleFrame – use standard frame for label

FUNCTION

Set this to `True` to have this label use a standard frame.

TYPE

Boolean

APPLICABILITY

I

23 Listtree class

23.1 Overview

Listtree class handles tree nodes which can be defined as node or as leaf. Only nodes can contain a list where other tree nodes can be inserted.

You can create a very complex tree. Two different lists of tree nodes exist: First the one which contains all the tree nodes you have inserted. All modifications are made to this list. The second list is the display list. As you can open or close a node, not all of the existing tree nodes are displayed.

The tree nodes can be inserted and removed, sorted, moved, exchanged or renamed. To sort you can also drag and drop them. Modifications can be made in relation to the whole tree, to one level, to a sub-tree or to only one tree node, the visibility is checked or not.

The user can control the listtree by the MUI keys, this means a node is opened with "Right" and closed with "Left". Check your MUI preferences for the specified keys.

Only one entry can be selected yet, so you cannot use the listtree for multi selecting.

If you do not set `Listtree.DragDropSort` to `False`, the list tree will become active for Drag&Drop. This means you can drag one entry and drop it on the same listtree again. While dragging an indicator shows where to drop. You cannot drop an entry on itself, nor you can drop an opened node on any of its members.

When creating a listtree in XML code, you can use Listtreenode class to fill it with nodes and items. Here is an example declaration:

```
<listtree>
   <node name="CPU">
      <item>Model: Motorola MPC 7447/7457 Apollo V1.1</item>
      <item>CPU speed: 999 Mhz</item>
      <item>FSB speed: 133 Mhz</item>
      <item>Extensions: performancemonitor altivec</item>
   </node>
   <node name="Machine">
      <item>Machine name: Pegasos II</item>
      <item>Memory: 524288 KB</item>
      <item>Extensions: bus.pci bus.agp</item>
   </node>
   <node name="Expansion buses">
      <node name="PCI/AGP">
         <item>Vendor 0x11AB Device 0x6460</item>
      </node>
   </node>
   <node name="Libraries">
      <item>0x6c7d4a58: exec.library V53.34</item>
   </node>
   <node name="Devices">
      <item>0x6ff8fba4: ramdrive.device V52.6</item>
   </node>
```

```
        <node name="Tasks">
           <node name="input.device">
              <item>Stack: 0x6ff4b000 - 0x6ff5b000</item>
              <item>Signals: SigWait 0x00000000</item>
              <item>State: Task (Waiting)</item>
           </node>
        </node>
     </listtree>
```

In this example we have made use of the `Listtreenode.Name` attribute to add a name to each of our nodes. There are some more attributes that you can use to customize the appearance of your nodes. See Section 24.1 [Listtreenode class], page 141, for details.

The strings you specify using the `<item>` tag and using the `Listtreenode.Name` attribute can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

23.2 Listtree.Active

NAME

Listtree.Active – set/get active tree entry

FUNCTION

Setting this attribute will move the cursor on the specified tree node, if it is visible. If the node is in an opened tree the listview is scrolling into the visible area. Passing the special value `Off` will vanish the cursor.

If this attribute is read it returns the id of the active tree node. The result is `Off` if there is no active entry.

You can create a notification on `Listtree.Active`. The `TriggerValue` is the active tree node.

TYPE

String

APPLICABILITY

SGN

23.3 Listtree.AutoLineHeight

NAME

Listtree.AutoLineHeight – enable automatic line height calculation (V1.4)

FUNCTION

Enables automatic line height calculation. Useful e.g. if you cannot predict a sensible line height.

This attribute requires MUI 4 or better. On older versions of MUI you can use the `Listtree.MinLineHeight` attribute to change the line height. See Section 23.12 [Listtree.MinLineHeight], page 135, for details.

TYPE

Boolean

APPLICABILITY

I

23.4 Listtree.Close

NAME

Listtree.Close – close list node

SYNOPSIS

```
mui.DoMethod(id, "Close", listnode$, treenode$)
```

FUNCTION

Closes a node or nodes of a listtree. When the active entry was a child of the closed node, the closed node will become active.

Listnode$ specifies the node whose list is used to find the entry. The search is started at the head of the list. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.
`Parent`	Parent list.

Treenode$ specifies the node which is to be closed. If there are children of the node, they are removed from the display list. This can be the string identifier of a node or one of the following special values:

`Head`	The head of the list defined in listnode$ is closed.
`Tail`	Closes the tail of the list.
`Active`	Closes the active node.
`All`	All nodes of the list which is specified in listnode$ are closed.

INPUTS

`id`	id of the listtree object
`listnode$`	id of list node to use or special value (see above)
`treenode$`	id of tree node to use or special value (see above)

23.5 Listtree.DoubleClick

NAME

Listtree.DoubleClick – get notified about double clicks on tree nodes

FUNCTION

You can setup a notification on this attribute to learn about double clicks on tree nodes. The tree node that the user double-clicked will be passed in `TriggerValue`.

TYPE

Boolean

APPLICABILITY

N

23.6 Listtree.DragDropSort

NAME

Listtree.DragDropSort – enable tree sorting via drag'n'drop

FUNCTION

Setting this attribute to `False` will disable the ability to sort the list tree by drag&drop. It is enabled by default.

TYPE

Boolean

APPLICABILITY

IS

23.7 Listtree.EmptyNodes

NAME

Listtree.EmptyNodes – hide list indicator for empty nodes

FUNCTION

Setting this attribute to `True` will display all empty nodes as leaves, this means no list indicator is shown. Nevertheless the entry is handled like a node.

TYPE

Boolean

APPLICABILITY

IS

23.8 Listtree.Exchange

NAME

Listtree.Exchange – exchange two tree nodes

SYNOPSIS

```
mui.DoMethod(id, "Exchange", listnd1$, treend1$, listnd2$, treend2$)
```

FUNCTION

Exchanges the two tree entries specified in listnd1$ and listnd2$.

Listnd1$ specifies the node whose list is used to find the entry. The search is started at the beginning of this list. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.

`Active`	The list of the active node.

Treend1$ specifies the node which is to be exchanged. This can be the string identifier of a node or one of the following special values:

`Head`	The head of the list defined in listnd1$ is exchanged.
`Tail`	Exchanges the tail of the list.
`Active`	Exchanges the active node.

Listnd2$ specifies the node whose list is used to find the entry that the first entry should be exchanged with. The search is started at the beginning of this list. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.

Treend2$ specifies the node which is to be exchanged. This can be the string identifier of a node or one of the following special values:

`Head`	The head of the list defined in listnd2$ is exchanged.
`Tail`	Exchanges the tail of the list.
`Active`	Exchanges the active node.
`Up`	Exchanges with the next entry.
`Down`	Exchanges with the previous entry.

INPUTS

`id`	id of the listtree object
`listnd1$`	id of list node to use or special value (see above)
`treend1$`	id of tree node to use or special value (see above)
`listnd2$`	id of list node to use or special value (see above)
`treend2$`	id of tree node to use or special value (see above)

23.9 Listtree.GetEntry

NAME

Listtree.GetEntry – get tree entry

SYNOPSIS

```
found, table = mui.DoMethod(id, "GetEntry", node, position, flags$)
```

FUNCTION

Get another node in relation to the specified list or node. This method allows you to traverse the entire list tree. See below for an example.

"Node" specifies the node which is used to find another one. This can also be a list node, if the position a related to a list. In contrast to all other methods or attributes of

Listtree class, "node" must NOT be a string identifier but a special value returned by this method or by `Listtree.FindName` in the `MuiID` field of the return table (see below). Alternatively, it can be one of the following special values:

`Root`	The root list is used.
`Active`	The list with the active entry is used.

Position specifies the number of nodes of the list specified in "node" or one of the following special values:

`Head`	The head of the list in "node" is returned.
`Tail`	The tail of the list is returned.
`Active`	The active node is returned.
`Next`	The next node after the tree node "node" is returned.
`Previous`	The node before the tree node "node".
`Parent`	The list node of the "node", it's the parent one.

Flags$ can be a combination of the following options:

`SameLevel`	Only nodes on the same level are affected.
`Visible`	The position is counted on visible entries only.

If you specify multiple of the flags above, you have to separate them using a semicolon, e.g. "SameLevel; Visible".

This method returns two values: The first return value is a boolean flag which indicates whether or not a node was found. If the first return value is `True`, the second return value is a table with the following fields initialized:

`Name`	Name of the tree node.
`Node`	`True` if the found entry is a node, `False` if it is a leaf.
`ID`	String object identifier of this tree node.
`MuiID`	Internal MUI ID of this tree node. This is the only id you are allowed to pass in the 'node' argument of this method. Passing standard string object identifiers is not allowed by this method. You can use this value for subsequent calls to `Listtree.GetEntry` in the "node" argument See above for more information and below for an example.

INPUTS

`id`	id of the listtree object
`node`	special node identifier returned by this method
`position`	index of entry to get
`flags$`	combination of flags to use (see above)

RESULTS

`found`	boolean flag indicating whether or not an entry was found

`table`	table containing information about found entry

EXAMPLE

```
Function p_DumpListTree(id$, node, indent)
   Local found, t = mui.DoMethod(id$, "GetEntry", node, "Head", "")
   While found = True
      If indent > 0
         DebugPrint(RepeatStr(" ", indent) ..
             IIf(t.Node = True, "+", "") .. t.name)
      Else
         DebugPrint(IIf(t.Node = True, "+", "") .. t.name)
      EndIf
      If t.Node = True Then p_DumpListTree(id$, t.muiid, indent + 4)
      found, t = mui.DoMethod(id$, "GetEntry", t.muiid, "Next", "")
    Wend
EndFunction

p_DumpListTree("mylisttree", "root", 0)
```

The code above shows how to dump the complete contents of a listtree, preserving its structure.

23.10 Listtree.FindName

NAME

Listtree.FindName – find node by name

SYNOPSIS

```
found, table = mui.DoMethod(id, "FindName", listnode$, name$, flags$)
```

FUNCTION

Find a node whose name matches the specified one. The search is done without paying attention to case sensitivity.

Listnode$ specifies the node whose list is used to find the name. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.

Flags$ can be a combination of the following options:

`SameLevel`	Only nodes on the same level are affected.
`Visible`	Only visible entries are taken into account.

If you specify multiple of the flags above, you have to separate them using a semicolon, e.g. "SameLevel; Visible".

This method returns two values: The first return value is a boolean flag which indicates whether or not a node was found. If the first return value is `True`, the second return value is a table with the following fields initialized:

`Name`	Name of the tree node. As the search is conducted in case insensitive mode, this item allows you to find out the real spelling of the entry.
`Node`	`True` if the found entry is a node, `False` if it is a leaf.
`ID`	String object identifier of this tree node.
`MuiID`	Internal MUI ID of this tree node. This can be used with `Listtree.GetEntry`.

INPUTS

`id`	id of the listtree object
`listnode$`	id of list node to use or special value (see above)
`name$`	name to look for
`flags$`	lookup flags (see above)

RESULTS

`found`	boolean flag indicating whether or not an entry was found
`table`	table containing information about found entry

23.11 Listtree.Insert

NAME

Listtree.Insert – insert new tree node

SYNOPSIS

```
mui.DoMethod(id, "Insert", entry$, id$, listnode$, prevnode$, flags$)
```

FUNCTION

Inserts entry$ at the position which is defined with listnode$ and prevnode$. Entry$ contains the name of the entry as string. id$ must be a unique string identifier that you want to use to refer to the newly inserted tree node.

In listnode$ you specify the node whose list is used to insert the entry. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.

In prevnode$ you have to specify the node which is the predecessor of the node to insert. This can be the string identifier of a node or one of the following special values:

`Head`	It will be inserted at the head of the list.
`Tail`	It will be inserted at the tail of the list.
`Active`	It will be inserted after the active node.
`Sorted`	The node is inserted using the sort hook.

Flags$ can be a combination of the following options:

`List`	The node contains a list where other nodes can be inserted.
`Open`	The list node is open, sub nodes are displayed.
`Frozen`	The node doesn't react on doubleclick or open/close by user.
`NoSign`	The indicator of list nodes isn't shown.
`Active`	The inserted entry will be set active, this means the cursor is moved on it.
`NextNode`	prevnode$ specifies the successor, not the predecessor, i.e. the node will be inserted before prevnode$, not after prevnode$.

If you specify multiple of the flags above, you have to separate them using a semicolon, e.g. "Active; NextNode".

INPUTS

`id`	id of the listtree object
`entry$`	name of the entry to insert
`id$`	unique string identifier for new tree node
`listnode$`	id of list node to use or special value (see above)
`prevnode$`	id of tree node to use or special value (see above)
`flags$`	insertion flags (see above)

23.12 Listtree.MinLineHeight

NAME

Listtree.MinLineHeight – set minimum line height (V1.4)

FUNCTION

Sets the minimum line height for the listtree in pixels. Useful e.g. if you have custom images.

Alternatively, you can also enable automatic line height calculation for the listtree using the `Listtree.AutoLineHeight` attribute. This is more convenient but requires at least MUI 4.0. See Section 23.3 [Listtree.AutoLineHeight], page 128, for details.

TYPE

Number

APPLICABILITY

I

23.13 Listtree.Move

NAME

Listtree.Move – move an entry to a new position

SYNOPSIS

```
mui.DoMethod(id, "Move", olistnode$, otreenode$, nlistnode$, ntreenode$)
```

FUNCTION

Move an entry to the position after a defined node.

Olistnode$ specifies the node which list is used to find the entry. The search is started at the head of the list. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.

Otreenode$ specifies the node which should be moved. This can be the string identifier of a node or one of the following special values:

`Head`	The head of the list defined in olistnode$ is moved.
`Tail`	The tail of the list is moved.
`Active`	The active node is moved.

Nlistnode$ specifies the node whose list is used to find the entry. The search is started at the head of the list. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.

Ntreenode$ specifies the node which is the predecessor of the entry which is inserted. This can be the string identifier of a node or one of the following special values:

`Head`	The node is moved to the head of the list defined in nlistnode$.
`Tail`	The node is moved to the tail of the list.
`Active`	The node is moved after the active node.
`Sorted`	The node is moved to the list using the sort hook.

INPUTS

`id`	id of the listtree object
`olistnode$`	id of list node to use or special value (see above)
`otreenode$`	id of tree node to use or special value (see above)
`nlistnode$`	id of list node to use or special value (see above)
`ntreenode$`	id of tree node to use or special value (see above)

23.14 Listtree.Open

NAME

Listtree.Open – open list node

SYNOPSIS

```
mui.DoMethod(id, "Open", listnode$, treenode$)
```

FUNCTION

Opens a node in the listtree. To open a child which isn't displayed use `Parent` as listnode$ to open all its parents, too. Only nodes can be opened.

Listnode$ specifies the node whose list is used to find the entry. The search is started at the head of the list. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.
`Parent`	Flag to open all the parents of the node, too.

Treenode$ specifies the node which is to be opened. This can be the string identifier of a node or one of the following special values:

`Head`	Opens the head node of the list.
`Tail`	Opens the tail node of the list.
`Active`	The active node will be opened.
`All`	All the nodes of the list are opened.

INPUTS

`id`	id of the listtree object
`listnode$`	id of list node to use or special value (see above)
`treenode$`	id of tree node to use or special value (see above)

23.15 Listtree.Quiet

NAME

Listtree.Quiet – disable listtree refresh (V1.4)

FUNCTION

If you add/remove lots of entries to/from a currently visible listtree, this will cause lots of screen action and slow down the operation. Setting `Listtree.Quiet` to `True` will temporarily prevent the listtree from being refreshed, this refresh will take place only once when you set it back to false again.

TYPE

Boolean

APPLICABILITY

S

23.16 Listtree.Remove

NAME

Listtree.Remove – remove tree node

SYNOPSIS

```
mui.DoMethod(id, "Remove", listnode$, treenode$)
```

FUNCTION

Removes a node or nodes from a listtree. When the active entry is removed, the following entry will become active.

Listnode$ specifies the node whose list is used to find the entry. The search is started at the beginning of this list. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.

Treenode$ specifies the node which is to be removed. If there are children of the node, they are also removed. This can be the string identifier of a node or one of the following special values:

`Head`	The head of the list defined in listnode$ is removed.
`Tail`	Removes the tail of the list.
`Active`	Removes the active node.
`All`	All nodes of the list which is specified in listnode$ are removed. Other nodes of parent lists are not affected.

INPUTS

`id`	id of the listtree object
`listnode$`	id of list node to use or special value (see above)
`treenode$`	id of tree node to use or special value (see above)

23.17 Listtree.Rename

NAME

Listtree.Rename – rename list node

SYNOPSIS

```
mui.DoMethod(id, "Rename", treenode$, newname$)
```

FUNCTION

Rename the name of specified node.

Treenode$ specifies the node which should be renamed. This can be the string identifier of a node or one of the following special values:

`Active`	The active tree node is used.

INPUTS

`id`	id of the listtree object
`treenode$`	id of tree node to use or special value (see above)
`newname$`	new name for entry

23.18 Listtree.Sort

NAME

Listtree.Sort – sort list node

SYNOPSIS

```
mui.DoMethod(id, "Sort", listnode$)
```

FUNCTION

Sorts a list node using the mode specified in `Listtree.SortHook`.

Listnode$ specifies the node whose list should be sorted. This can be the string identifier of a node or one of the following special values:

`Root`	The root list.
`Active`	The list of the active node.

INPUTS

`id`	id of the listtree object
`listnode$`	id of list node to use or special value (see above)

23.19 Listtree.SortHook

NAME

Listtree.SortHook – set sort mode

FUNCTION

Set this attribute to the desired sort mode. When you are using `Listtree.Insert` with `Sorted` or dropping an entry on a closed node, this sort hook is used.

The following values are possible here:

`Head`	Any entry is inserted at head of the list.
`Tail`	Any entry is inserted at tail of the list.

`LeavesTop`
: Leaves are inserted at top of the list, nodes at bottom. They are alphabetically sorted.

`LeavesMixed`
: The entries are only alphabetically sorted.

`LeavesBottom`
: Leaves are inserted at bottom of the list, nodes at top. They are alphabetically sorted. This is the default.

TYPE

String (see above for possible values)

APPLICABILITY

IS

23.20 Listtree.Title

NAME

Listtree.Title – set listtree title

FUNCTION

Specify a title for the current listtree.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

I

24 Listtreenode class

24.1 Overview

Listtreenode class is needed when creating listtrees. It allows you to define the single nodes of the listtree and specify different attributes for them.

Listtreenode class must always be embedded inside a `<listtree>` declaration. See Section 23.1 [Listtree class], page 127, for details.

24.2 Listtreenode.Frozen

NAME

Listtreenode.Frozen – create a dead node

FUNCTION

If you set this to `True`, the node will not react on doubleclick or open/close by the user.

TYPE

Boolean

APPLICABILITY

I

24.3 Listtreenode.Name

NAME

Listtreenode.Name – set node name

FUNCTION

Specifies a string that should be used as the node name. This attribute is mandatory and must always be set.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

I

24.4 Listtreenode.NoSign

NAME

Listtreenode.NoSign – hide list indicator

FUNCTION

If you set this to `True`, the indicator of list nodes won't be shown.

TYPE

Boolean

APPLICABILITY

I

24.5 Listtreenode.Open

NAME

Listtreenode.Open – set node state

FUNCTION

Set this to `True` to open the node by default. All sub nodes are displayed then.

TYPE

Boolean

APPLICABILITY

I

25 Listview class

25.1 Overview

This class creates a listview. MUI's listview class is very powerful. It handles various types of entries and multi column lists are also supported, the format for a column is adjustable.

Listviews support any kind of sorting, multi selection and an active entry that can be controlled with the mouse or the cursor keys.

When creating a listview in XML code, you always have to add at least one column to it. This is done by using the Listviewcolumn class. Here is an example of a minimal listview declaration with just a single column:

```
<listview>
   <column/>
</listview>
```

It is also possible to add some entries to the listview right at declaration time. This can be done by using the `<item>` tag:

```
<listview>
   <column>
      <item>Entry 1</item>
      <item>Entry 2</item>
      <item>Entry 3</item>
   </column>
</listview>
```

The strings you specify using the `<item>` tag can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

If you want to have a multi-column list, you need to use the `<column>` tag several times. Here is an example:

```
<listview>
   <column title="Column 1">
      <item>Entry 1</item>
      <item>Entry 2</item>
      <item>Entry 3</item>
   </column>
   <column title="Column 2">
      <item>Entry 1</item>
      <item>Entry 2</item>
      <item>Entry 3</item>
   </column>
   <column title="Column 3">
      <item>Entry 1</item>
      <item>Entry 2</item>
      <item>Entry 3</item>
   </column>
</listview>
```

In this example we have also made use of the `Listviewcolumn.Title` attribute to add a title bar to each of our columns. There are some more attributes that you can use to customize the appearance of your columns. See Section 26.1 [Listviewcolumn class], page 159, for details.

25.2 Listview.Active

NAME

Listview.Active – set/get active list entry

FUNCTION

Reading this attribute will return the number of the active entry (the one with the cursor on it). The result is between 0 and `Listview.Entries`-1 or `Off`, in which case there is currently no active entry.

Setting the attribute will cause the list to move the cursor to the new position and scroll this position into the visible area.

The following special values are possible when setting this attribute:

`Off`	Clear selection.
`Top`	Select top entry.
`Bottom`	Select bottom entry.
`Up`	Select previous entry.
`Down`	Select next entry.
`PageUp`	Move list cursor one page up.
`PageDown`	Move list cursor one page down.

TYPE

Number or string (see above for possible values)

APPLICABILITY

ISGN

25.3 Listview.AdjustHeight

NAME

Listview.AdjustHeight – fix listview height

FUNCTION

A list with `Listview.AdjustHeight` set to true is exactly as high as all of its entries and not resizable. This is only possible when the list is filled *before* the window is opened.

TYPE

Boolean

APPLICABILITY

I

25.4 Listview.AdjustWidth

NAME

Listview.AdjustWidth – fix listview width

FUNCTION

A list with `Listview.AdjustWidth` set to true is exactly as wide as the widest entry and not resizable. This is only possible when the list is filled *before* the window is opened.

TYPE

Boolean

APPLICABILITY

I

25.5 Listview.AutoLineHeight

NAME

Listview.AutoLineHeight – enable automatic line height calculation (V1.4)

FUNCTION

Enables automatic line height calculation. Useful e.g. if you cannot predict a sensible line height.

This attribute requires MUI 4 or better. On older versions of MUI you can use the `Listview.MinLineHeight` attribute to change the line height. See Section 25.20 [Listview.MinLineHeight], page 151, for details.

TYPE

Boolean

APPLICABILITY

I

25.6 Listview.AutoVisible

NAME

Listview.AutoVisible – automatically jump to active entry

FUNCTION

Set this to make your lists automatically jump to the active entry when they are displayed.

TYPE

Boolean

APPLICABILITY

ISG

25.7 Listview.Clear

NAME

Listview.Clear – clear listview

SYNOPSIS

`mui.DoMethod(id, "Clear")`

FUNCTION

Clear the list, all entries are removed.

INPUTS

`id`	id of the listview object

25.8 Listview.ClickColumn

NAME

Listview.ClickColumn – learn about column clicks

FUNCTION

When using a multi column list, this attribute contains the number of the column where the user clicked.

TYPE

Number

APPLICABILITY

GN

25.9 Listview.DefClickColumn

NAME

Listview.DefClickColumn – set default column

FUNCTION

When the listview is controlled with the keyboard and the user presses RETURN, the value given here will be used as default for `Listview.ClickColumn`.

TYPE

Number

APPLICABILITY

ISG

25.10 Listview.DoubleClick

NAME

Listview.DoubleClick – learn about double click on listview

FUNCTION

This attribute is set to `True` whenever the user double clicks on an entry in the list.

TYPE

Boolean

APPLICABILITY

GN

25.11 Listview.Entries

NAME

Listview.Entries – get listview entries

FUNCTION

Get the current number of entries in the list.

TYPE

Number

APPLICABILITY

G

25.12 Listview.Exchange

NAME

Listview.Exchange – exchange two entries

SYNOPSIS

```
mui.DoMethod(id, "Exchange", pos1, pos2)
```

FUNCTION

Exchange two entries in a list. Positions have to be passed as absolute values starting from 0 to `Listview.Entries`-1 or pass one of the following special values:

`Top`	Use top entry.
`Active`	Use active entry.
`Bottom`	Use bottom entry.
`Next`	Use next entry. This is only valid for the second parameter.
`Previous`	Use previous entry. This is only valid for the second parameter.

INPUTS

`id`	id of the listview object
`pos1`	number of the first entry
`pos2`	number of the second entry

25.13 Listview.First

NAME

Listview.First – get first visible entry

FUNCTION

Get the number of the entry displayed on top of the list. You have to be prepared to get a result of -1, which means that the list is not visible at all (e.g. when the window is iconifed).

TYPE

Number

APPLICABILITY

G

25.14 Listview.GetEntry

NAME

Listview.GetEntry – get listview entry

SYNOPSIS

```
column1$, ... = mui.DoMethod(id, "GetEntry", pos)
```

FUNCTION

Get an entry of a list. Pass `Active` in pos to get the active entry. `Listview.GetEntry` will return the entries of all columns in the row specified by pos. You will get as many return values as there are columns in the listview.

INPUTS

`id`	id of the listview object
`pos`	index of listview row or "Active"

RESULTS

`column1$`	entry data of first column
`...`	further data if listview has multiple columns

25.15 Listview.GetSelection

NAME

Listview.GetSelection – get selected entries

SYNOPSIS

```
t = mui.DoMethod(id, "GetSelection")
```

FUNCTION

This method steps through the contents of a (multi select) listview and returns every entry that is currently selected. When no entry is selected but an entry is active, only the active entry will be returned.

This behaviour will result in not returning the active entry when you have some other selected entries somewhere in your list. Since the active entry just acts as some kind of cursor mark, this seems to be the only sensible possibility to handle multi selection together with keyboard control.

INPUTS

`id`	id of the listview object

RESULTS

`t`	table containing selected entries

25.16 Listview.Input

NAME

Listview.Input – configure input mode of listview

FUNCTION

Setting this to `False` will result in a read only list view. Defaults to `True`.

TYPE

Boolean

APPLICABILITY

I

25.17 Listview.Insert

NAME

Listview.Insert – insert new entry

SYNOPSIS

```
mui.DoMethod(id, "Insert", pos, column1$, ...)
```

FUNCTION

Insert one new entry into a list. If the listview has multiple columns, you need to pass as many arguments as there are columns in the listview.

The insert position is specified in the "pos" argument. The new entry will be added in front of the entry specified by "pos". This can be an absolute index position starting at 0 for the first entry or one of the following special values:

`Top`	Insert as first entry.
`Active`	Insert in front of the active entry.
`Sorted`	Insert sorted.
`Bottom`	Insert as last entry.

If "pos" is bigger or equal to the number of entries in the list, it's treated like `Bottom`.

All entries you insert using this method can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

INPUTS

`id`	id of the listview object
`pos`	insert position as absolute number or special value (see above)
`column1$`	entry to insert into first column
`...`	more entries if listview has multiple columns

25.18 Listview.InsertPosition

NAME

Listview.InsertPosition – query position of newly inserted entry

FUNCTION

After insertion of an element with `Listview.Insert`, you can query the position of the new entry by getting this attribute.

TYPE

Number

APPLICABILITY

G

25.19 Listview.Jump

NAME

Listview.Jump – scroll to an entry

SYNOPSIS

```
mui.DoMethod(id, "Jump", pos)
```

FUNCTION

Scroll any entry into the visible part of a list.

Note: Jumping to an entry doesn't mean to make this entry the active one. This can be done by setting the `Listview.Active` attribute.

Starting with MUI Royale 1.4, "pos" can also be one of the following special values:

`Active`	Jump to active item.
`Top`	Jump to top.
`Bottom`	Jump to bottom.
`Up`	Jump one page up.
`Down`	Jump one page down.

INPUTS

`id`	id of the listview object
`pos`	number of the entry that should be made visible. Pass "Active" to jump to the active entry.

25.20 Listview.MinLineHeight

NAME

Listview.MinLineHeight – set minimum line height

FUNCTION

Sets the minimum line height for lists in pixels. Useful e.g. if you have custom images.

Alternatively, you can also enable automatic line height calculation for the listview using the `Listview.AutoLineHeight` attribute. This is more convenient but requires at least MUI 4.0. See Section 25.5 [Listview.AutoLineHeight], page 145, for details.

TYPE

Number

APPLICABILITY

I

25.21 Listview.Move

NAME

Listview.Move – move entry to new position

SYNOPSIS

```
mui.DoMethod(id, "Move", from, to)
```

FUNCTION

Move an entry from one position to another. Positions have to be passed as absolute values starting from 0 to `Listview.Entries`-1 or pass one of the following special values:

`Top`	Use top entry.
`Active`	Use active entry.
`Bottom`	Use bottom entry.
`Next`	Use next entry. This is only valid for the second parameter.
`Previous`	Use previous entry. This is only valid for the second parameter.

INPUTS

`id`	id of the listview object
`from`	number of the first entry
`to`	number of the second entry

25.22 Listview.MultiSelect

NAME

Listview.MultiSelect – set multi select mode for listview

FUNCTION

Four possibilities exist for a listviews multi select capabilities:

`None`	The listview cannot multiselect at all.
`Default`	The multi select type (with or without shift) depends on the users preferences setting.
`Shifted`	Overrides the users prefs, multi selecting only together with shift key.
`Always`	Overrides the users prefs, multi selecting without shift key.

Please do *not* override the users prefs unless you have a good reason!

TYPE

String (see above for possible values)

APPLICABILITY

I

25.23 Listview.Quiet

NAME

Listview.Quiet – disable listview refresh (V1.2)

FUNCTION

If you add/remove lots of entries to/from a currently visible list, this will cause lots of screen action and slow down the operation. Setting `Listview.Quiet` to `True` will temporarily prevent the list from being refreshed, this refresh will take place only once when you set it back to false again.

TYPE

Boolean

APPLICABILITY

S

25.24 Listview.Remove

NAME

Listview.Remove – remove entry from list

SYNOPSIS

```
mui.DoMethod(id, "Remove", pos)
```

FUNCTION

Remove an entry from a list. The position can be specified as an absolute index value or as one of the following special values:

`First`	Remove first entry.
`Active`	Remove active entry.

`Last`	Remove last entry.
`Selected`	Remove selected entry.

When the active entry is removed, the following entry will become active.

INPUTS

`id`	id of the listview object
`pos`	index of entry to remove or one of the special values (see above)

25.25 Listview.Rename

NAME

Listview.Rename – rename an entry (V1.1)

SYNOPSIS

```
mui.DoMethod(id, "Rename", pos, column1$, ...)
```

FUNCTION

Renames the listview entry at the specified position. If the listview has multiple columns, you need to pass as many arguments as there are columns in the listview. It is not possible to rename only a single column entry - this method always affects the complete row so you need to pass as many strings as there are columns in your listview.

The entry position is specified in the `"pos"` argument. This can be an absolute index position starting at 0 for the first entry or one of the following special values:

`Active`	Rename the active entry.

All strings you pass to this method can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

INPUTS

`id`	id of the listview object
`pos`	entry position as absolute number or special value (see above)
`column1$`	new text for entry in the first column
`...`	more entries if listview has multiple columns

25.26 Listview.Select

NAME

Listview.Select – (de)select list entry

SYNOPSIS

```
mui.DoMethod(id, "Select", pos, seltype)
```

FUNCTION

Select/deselect a list entry or ask an entry if its selected.

`"Pos"` can be either the number of the entry or one of the following special values:

`Active`	For the active entry.

`All`	For all entries.

"Seltype" can be one of the following:

`Off`	Unselect entry.
`On`	Select entry.
`Toggle`	Toggle entry.

INPUTS

`id`	id of the listview object
`pos`	entry index or special value (see above)
`seltype`	selection type (see above)

25.27 Listview.ScrollerPos

NAME

Listview.ScrollerPos – set scroller position for listview

FUNCTION

Specifies the position of a listviews scrollbar. Don't use this tag unless it's absolutely required!

If you specify `None` here, your listview won't get a scroller at all and look much like a list object alone. However, listviews without scroller are still more powerful than list objects as they feature e.g. drag&drop possibilities.

Creating listviews without a scrollbar makes sense if you want to have the scrollbar somewhere else, e.g. outside of a horizontal virtual group where the listview resides. This technique allows the creation of horizontally scrollable listviews.

The following options are possible:

`Default`	Position scrollbar according to user preferences.
`Left`	Position scrollbar to the left of the listview.
`Right`	Position scrollbar to the right of the listview.
`None`	No scrollbar.

TYPE

String (see above for possible values)

APPLICABILITY

I

25.28 Listview.SelectChange

NAME

Listview.SelectChange – learn about selection changes

FUNCTION

This attribute is set to `True` whenever the selection state of one or more items in the list is changing. You can use this e.g. if you want to display the number of selected items in a status line.

TYPE

Boolean

APPLICABILITY

GN

25.29 Listview.Sort

NAME

Listview.Sort – sort the list

SYNOPSIS

```
mui.DoMethod(id, "Sort", column)
```

FUNCTION

Sort the list. MUI uses an iterative quicksort algorithm, no stack problems will occur. You have to pass the index of the column by which the list should be sorted in the third argument. Columns are counted from 0 to the number of columns minus 1, i.e. to sort the list by the first column pass 0 as the column index.

INPUTS

`id`	id of the listview object
`column`	index of column to use sort list by (V1.1)

25.30 Listview.SortFunc

NAME

Listview.SortFunc – determine how entries should be sorted (V1.2)

FUNCTION

This attribute can be used to set up a callback function that will be called whenever MUI Royale needs to sort the list entries. The callback function will receive two entries as arguments and it has to determine which entry should be put first.

The callback function you specify here will be called like a standard MUI Royale event callback, but with the following extra arguments:

`Entry1:`	This is a table containing all elements of a single listview row. As listviews can have more than one column, this table contains as many string elements as there are columns in your listview.

`Entry2:` This is a table containing all elements of a single listview row. As listviews can have more than one column, this table contains as many string elements as there are columns in your listview.

`SortColumn:`
This contains the column index by which the list should be sorted. Columns are counted from 0 to the number of columns minus 1. See Section 25.29 [Listview.Sort], page 155, for details.

Your callback function then has to return a value that indicates how the two entries should be aligned in the listview. If entry 1 should be placed before 2, your callback has to return -1. If entry 1 should be placed after entry 2, your callback has to return 1. If the two entries are the same, return 0.

Note that you always have to pass a string specifying the name of a Hollywood function to this attribute. Never pass the function directly but always pass the name of the function as a string!

Also note that you must also set up a notification on this attribute. Otherwise the callback function will never get called.

See Section 3.6 [Notifications], page 12, for details.

TYPE
String

APPLICABILITY
ISGN

25.31 Listview.TitleClick

NAME
Listview.TitleClick – learn about title clicks (V1.1)

FUNCTION
This attribute is set to the column index number whenever the user clicks on a column title button.

This attribute is only supported on AmigaOS 4 and MorphOS.

TYPE
Number

APPLICABILITY
N

25.32 Listview.Visible

NAME
Listview.Visible – get number of visible entries

FUNCTION

Get the current number of visible entries in the list. You have to be prepared to get a result of -1, which means that the list is not visible at all (e.g. when the window is iconifed).

TYPE

Number

APPLICABILITY

G

26 Listviewcolumn class

26.1 Overview

Listviewcolumn class is needed when creating listviews. It allows you to specify different attributes for the columns of your listviews.

Listviewcolumn class must always be embedded inside a `<listview>` declaration. See Section 25.1 [Listview class], page 143, for details.

26.2 Listviewcolumn.Bar

NAME

Listviewcolumn.Bar – use separator bar between columns

FUNCTION

Since muimaster.library V11, you can enable a vertical bar between this and the next column by using this switch.

TYPE

Boolean

APPLICABILITY

I

26.3 Listviewcolumn.Col

NAME

Listviewcolumn.Col – set column order

FUNCTION

This value adjusts the number of the current column. This allows you to adjust the order of your columns without having to change your display hook.

Defaults to current entry number (0,1,...).

This attribute is especially useful in connection with Dirlist class and Volumelist class to change the display order of the different attributes. See Section 14.1 [Dirlist class], page 87, for details.

Starting with MUI Royale 1.1 this attribute has an applicability of ISG. In MUI Royale 1.0 the applicability of this attribute was only I.

TYPE

Number

APPLICABILITY

ISG

26.4 Listviewcolumn.Delta

NAME

Listviewcolumn.Delta – set column width

FUNCTION

Space in pixel between this column and the next. the last displayed column ignores this setting. Defaults to 4.

Starting with MUI Royale 1.1 this attribute has an applicability of ISG. In MUI Royale 1.0 the applicability of this attribute was only I.

TYPE

Number

APPLICABILITY

ISG

26.5 Listviewcolumn.Hidden

NAME

Listviewcolumn.Hidden – show/hide listview column (V1.1)

FUNCTION

This attribute allows you to show or hide single listview columns. Note that the columns will still be there, they'll just be invisible. Thus, you must not forget hidden columns when changing listview entries using `Listview.Insert` or similar methods.

This attribute is only supported on AmigaOS 4 and MorphOS.

TYPE

Boolean

APPLICABILITY

ISG

26.6 Listviewcolumn.MinWidth

NAME

Listviewcolumn.MinWidth – set minimum column width

FUNCTION

Minimum percentage width for the current column. If your list is 200 pixel wide and you set this to 25, your column will at least be 50 pixel. The special value -1 for this parameter means that the minimum width is as wide as the widest entry in this column. This ensures that every entry will be completely visible (as long as the list is wide enough). Defaults to -1.

Starting with MUI Royale 1.1 this attribute has an applicability of ISG. In MUI Royale 1.0 the applicability of this attribute was only I.

TYPE

Number

APPLICABILITY
ISG

26.7 Listviewcolumn.MaxWidth

NAME
Listviewcolumn.MaxWidth – set maximum column width

FUNCTION
Maximum percentage width for the current column. If your list is 200 pixel wide and you set this to 25, your column will not be wider as 50 pixel. The special value -1 for this parameter means that the maximum width is as wide as the widest entry in this column. Defaults to -1.

Starting with MUI Royale 1.1 this attribute has an applicability of ISG. In MUI Royale 1.0 the applicability of this attribute was only I.

TYPE
Number

APPLICABILITY
ISG

26.8 Listviewcolumn.PreParse

NAME
Listviewcolumn.PreParse – set preparse text string for column

FUNCTION
A preparse value for this column. Setting this e.g. to `"\33c"` would make the column centered.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

Starting with MUI Royale 1.1 this attribute has an applicability of ISG. In MUI Royale 1.0 the applicability of this attribute was only I.

TYPE
String

APPLICABILITY
ISG

26.9 Listviewcolumn.Title

NAME
Listviewcolumn.Title – set column title

FUNCTION

Specify a title for the current column. The title is displayed at the very first line and doesn't scroll away when the list top position moves.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

Starting with MUI Royale 1.1 this attribute has an applicability of ISG. In MUI Royale 1.0 the applicability of this attribute was only I.

TYPE

String

APPLICABILITY

ISG

26.10 Listviewcolumn.Weight

NAME

Listviewcolumn.Weight – set column weight

FUNCTION

The weight of the column. As with MUI's group class, columns are layouted with a minimum size, a maximum size and weight. A column with a weight of 200 would gain twice the space than a column with a weight of 100. Defaults to 100.

Starting with MUI Royale 1.1 this attribute has an applicability of ISG. In MUI Royale 1.0 the applicability of this attribute was only I.

TYPE

Number

APPLICABILITY

ISG

27 Menu class

27.1 Overview

Objects of menu class describe exactly one pulldown menu. They don't feature many options themselves, but as a subclass of Family class, they act as father for their several menu item objects which you can add to the menu using the Menuitem class. See Section 28.1 [Menuitem class], page 167, for details.

Menus must always be embedded inside a `<menustrip>` object. See Section 29.1 [Menustrip class], page 171, for details.

27.2 Menu.AddHead

NAME

Menu.AddHead – add detached object as first family child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddHead", obj)
```

FUNCTION

This method can be used to add the detached object specified by "obj" to the family object specified by "id". The detached object will be added as the family's first child. After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Menu.Remove` method.

INPUTS

`id`	id of the family object
`obj`	id of the object to attach

27.3 Menu.AddTail

NAME

Menu.AddTail – add detached object as last family child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddTail", obj)
```

FUNCTION

This method can be used to add the detached object specified by "obj" to the family object specified by "id". The detached object will be added as the family's last child. After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Menu.Remove` method.

INPUTS

`id`	id of the family object
`obj`	id of the object to attach

27.4 Menu.Disabled

NAME

Menu.Disabled – set/get disabled state of menu

FUNCTION

Enable or disable the complete menu.

TYPE

Boolean

APPLICABILITY

ISG

27.5 Menu.Insert

NAME

Menu.Insert – insert detached object after specified child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "Insert", obj, pred)
```

FUNCTION

This method can be used to insert the detached object specified by "obj" to the family object specified by "id". The detached object will be added after the child specified by "pred". After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Menu.Remove` method.

INPUTS

`id`	id of the family object
`obj`	id of the object to insert
`pred`	the object will be inserted after this object

27.6 Menu.Remove

NAME

Menu.Remove – detach object from family (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "Remove", obj)
```

FUNCTION

This method can be used to detach the specified object from the specified family. After this method returns the specified object will change its state from attached to detached. This means that you could now attach it to another family using a function like `Menu.Insert` or you could free it using `mui.FreeObject()`.

INPUTS

`id`	id of the family object
`obj`	id of the object to remove

27.7 Menu.Title

NAME

Menu.Title – set/get menu title

FUNCTION

Describe the title of the menu.

TYPE

String

APPLICABILITY

ISG

28 Menuitem class

28.1 Overview

Menuitem class describes a single menu item. You can use all of the gadtools menus features expect image menus here.

Since Menuitem class is a subclass of Family class, you can add other menu items as children of a menu item to indicate sub menus. MUI does not limit the level of sub menus, but the operating system currently allows a maximum nesting level of one. Because of this, children of menu items should not contain other menu items for now, the results are unpredictable.

Menuitems must always be embedded inside a `<menu>` tag which in turn have to be embedded inside a `<menustrip>` tag. See Section 29.1 [Menustrip class], page 171, for details.

Please note that the XML tag for menuitem class is just `<item>` and not `<menuitem>`.

See Section 29.1 [Menustrip class], page 171, for an example.

28.2 Menuitem.CommandString

NAME

Menuitem.CommandString – set/get command string

FUNCTION

Set to `True` if `Menuitem.Shortcut` points to a command string (e.g. "shift alt q") instead of a simple letter. Note that MUI won't check if these keys are pressed (just like intuition), you'll have to do this yourself.

TYPE

Boolean

APPLICABILITY

ISG

28.3 Menuitem.Disabled

NAME

Menuitem.Disabled – set/get disabled state of menu item

FUNCTION

Enable/disable the menu item.

TYPE

Boolean

APPLICABILITY

ISG

28.4 Menuitem.Exclude

NAME
Menuitem.Exclude – set/get exclude mask for radio menu items

FUNCTION
Bitmask of menu item numbers that are to be deselected when this one is selected.

TYPE
Number

APPLICABILITY
ISG

28.5 Menuitem.Selected

NAME
Menuitem.Selected – handle menu item selection state

FUNCTION
Set/get the selected state of a menu item. By setting up notification on this attribute, you can react on menu actions immediately.

TYPE
Boolean

APPLICABILITY
ISGN

28.6 Menuitem.Shortcut

NAME
Menuitem.Shortcut – set/get shortcut for menu item

FUNCTION
Define the shortcut for a menu item.

TYPE
String

APPLICABILITY
ISG

28.7 Menuitem.Title

NAME
Menuitem.Title – set/get menu item title

FUNCTION

Define the items title.

If MUI Royale 1.3 and MUI 4.0 or better is installed and this menu item is to appear in a context menu, you may use text formatting codes here. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

ISG

28.8 Menuitem.Type

NAME

Menuitem.Type – set/get menu item type

FUNCTION

This attribute can be used to set the type for this menu item. The following types are currently possible:

`Normal`	Normal menu item.
`Toggle`	Toggle menu item with a checkmark.
`Radio`	Menu item is part of a group of radio items. `Menuitem.Exclude` can be used to define the possible combinations of the radio items.

TYPE

String (see above for possible values)

APPLICABILITY

IG

29 Menustrip class

29.1 Overview

Menustrip class is the base class for MUI's object oriented menus. Its children are objects of Menu class, each of them describes exactly one menu.

A Menustrip object doesn't feature many options itself, but as a subclass of Family class, it simply acts as father for multiple Menu objects.

The Menustrip object is usually specified as a child of either Application class or window class with the attributes `Application.Menustrip` or `Window.Menustrip`. Additionally, it can also be used as a context menu for other MUI objects using the `Area.ContextMenu` attribute.

In an XML file a menu tree is defined using the `<menustrip>`, `<menu>` and `<item>` tags. Here is an example definition of a simple menustrip:

```
<menustrip id="mymenustrip">
    <menu title="Project">
        <item>New...</item>
        <item>Open...</item>
        <item/>
        <item>Save</item>
        <item>Save as...</item>
        <item/>
        <item>Quit</item>
    </menu>
    <menu title="Edit">
        <item shortcut="X">Cut</item>
        <item shortcut="C">Copy</item>
        <item shortcut="V">Paste</item>
    </menu>
    <menu title="?">
        <item>Settings...</item>
        <item>MUI settings...</item>
        <item/>
        <item>About...</item>
        <item>About MUI...</item>
    </menu>
</menustrip>
```

Note the empty `<item/>` declarations: These will insert a separator bar to the menu tree. Using separator bars makes your menu more readable to the end-user. After you have written the XML declaration above you can add the menustrip to one of your windows by using the `Window.Menustrip` attribute as follows:

```
<window menustrip="mymenustrip">
...
</window>
```

It is also possible to add a menustrip as a context menu to one of your window's gadgets using the `Area.ContextMenu` attribute. The context menu will then appear whenever the user presses the right mouse button over the gadget that has a context menu. Here is an example of how to add a menustrip as a context menu to a listview:

```
<listview contextmenu="mymenustrip">
...
</listview>
```

Please note that menustrips which are used as context menus must not contain more than one `<menu>` tree.

It is very important to note that you have to declare your menustrips in the `<application>` scope because menustrips are global objects and are only attached to windows or gadgets later on. That is why it is not allowed to declare menustrips inside a `<window>` XML scope.

29.2 Menustrip.AddHead

NAME

Menustrip.AddHead – add detached object as first family child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddHead", obj)
```

FUNCTION

This method can be used to add the detached object specified by `"obj"` to the family object specified by `"id"`. The detached object will be added as the family's first child. After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Menustrip.Remove` method.

INPUTS

`id`	id of the family object
`obj`	id of the object to attach

29.3 Menustrip.AddTail

NAME

Menustrip.AddTail – add detached object as last family child (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddTail", obj)
```

FUNCTION

This method can be used to add the detached object specified by `"obj"` to the family object specified by `"id"`. The detached object will be added as the family's last child. After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Menustrip.Remove` method.

INPUTS

`id`	id of the family object
`obj`	id of the object to attach

29.4 Menustrip.Insert

NAME

Menustrip.Insert – insert detached object after specified child

SYNOPSIS

```
mui.DoMethod(id, "Insert", obj, pred)
```

FUNCTION

This method can be used to insert the detached object specified by "obj" to the family object specified by "id". The detached object will be added after the child specified by "pred". After this method returns the specified object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Menustrip.Remove` method.

INPUTS

`id`	id of the family object
`obj`	id of the object to insert
`pred`	the object will be inserted after this object

29.5 Menustrip.Remove

NAME

Menustrip.Remove – detach object from family

SYNOPSIS

```
mui.DoMethod(id, "Remove", obj)
```

FUNCTION

This method can be used to detach the specified object from the specified family. After this method returns the specified object will change its state from attached to detached. This means that you could now attach it to another family using a function like `Menustrip.Insert` or you could free it using `mui.FreeObject()`.

INPUTS

`id`	id of the family object
`obj`	id of the object to remove

30 Notify class

30.1 Overview

Notify class is superclass of all other MUI classes. Its main purpose is to handle MUI's notification mechanism, but it also contains some other methods and attributes useful for every object.

Because Notify class is the super class for all other MUI classes, you can use all of its attributes with all other MUI classes. For example, you can use `Notify.HelpNode` to add online help reference to MUI gadgets or you could use `Notify.AppMessage` to add MUI objects to the app message handler.

30.2 Notify.AppMessage

NAME

Notify.AppMessage – get app message

FUNCTION

When your window is an AppWindow, i.e. you have set the `Window.AppWindow` attribute to `True`, you will be able to get AppMessages by listening to `Notify.AppMessage`. Whenever an AppMessage arrives, this attribute will trigger a notification.

For AppMessage notifications, the standard MUI Royale event handler callback will get two additional entries in the table that is passed as the parameter to your event handling function:

`NumDropFiles:`
The number of files the user dropped over your object. This is usually 1.

`DropFiles:`
A table containing the list of files that were dropped over the object. This table will have exactly 'NumDropFiles' entries.

See Section 3.6 [Notifications], page 12, for details.

`Notify.AppMessage` is object specific. You can e.g. set up different notifications for different objects in your window, they will only get executed when icons are dropped over the specific object.

If you wait on `Notify.AppMessage` with a window object, your notify will always get executed when icons are dropped on the window.

Note that AppWindows are only possible on the Workbench screen.

TYPE

String

APPLICABILITY

N

30.3 Notify.Class

NAME
Notify.Class – get class name of object

FUNCTION
Get the MUI Royale class name of an object.

TYPE
String

APPLICABILITY
G

30.4 Notify.ExportID

NAME
Notify.ExportID – set ID for settings export (V1.7)

FUNCTION
Objects with a `Notify.ExportID` export their contents during `Application.Save` and import them during `Application.Load`.

You have to use different export IDs for your objects!

TYPE
Four character string

APPLICABILITY
ISG

30.5 Notify.HelpLine

NAME
Notify.HelpLine – define line in help file

FUNCTION
Define a line in a help file specified with `Application.HelpFile`.

See Section 3.10 [Implementing online help], page 15, for details.

TYPE
Number

APPLICABILITY
ISG

30.6 Notify.HelpNode

NAME

Notify.HelpNode – define node in help file

FUNCTION

Define a node in a help file specified with `Application.HelpFile`.

See Section 3.10 [Implementing online help], page 15, for details.

TYPE

String

APPLICABILITY

ISG

30.7 Notify.ID

NAME

Notify.ID – set object ID

FUNCTION

This attribute can be used to set the ID for a MUI object. You need to give your objects unique IDs so that you can access them using the `mui.Set()`, `mui.Get()` and `mui.DoMethod()` functions.

TYPE

String

APPLICABILITY

I

30.8 Notify.MUIClass

NAME

Notify.MUIClass – get MUI class name of object

FUNCTION

Get the MUI class name of an object.

TYPE

String

APPLICABILITY

G

30.9 Notify.NoNotify

NAME

Notify.NoNotify – disable notifications

FUNCTION

If you set up a notify on an attibute to react on user input, you will also recognize events when you change this attribute under program control with `mui.Set()`. Setting `Notify.NoNotify` together with your attribute will prevent this notification from being triggered.

`Notify.NoNotify` is a "one time" attribute. It's only valid during the current `mui.Set()` call!

TYPE

Boolean

APPLICABILITY

S

30.10 Notify.NotifyData

NAME

Notify.NotifyData – set/get event specific user data

FUNCTION

This attribute allows you to define notification specific user data in an object. You have to pass a string here that contains one or more notifications and user data for each notification in the string. When a notification that is specified in the string is triggered, the event handler callback will receive the user data specified in `Notify.NotifyData` in the `NotifyData` field of the event message.

The string that you need to pass to this attribute must be formatted as follows: Name of the notification attribute, followed by a colon, followed by a user data string, followed by a semi-colon. The sequence may then be repeated as many times as it is requred.

For example: "Active: foo; DoubleClick: bar;". When the "Active" notification is triggered, "foo" will be send to the event handler callback. When the "DoubleClick" attribute triggers, "bar" will be sent.

See Section 3.6 [Notifications], page 12, for details.

TYPE

Any

APPLICABILITY

ISG

30.11 Notify.Revision

NAME

Notify.Revision – get revision number of object class

FUNCTION

Get the revision number of an objects class. Although `Notify.Revision` is documented at notify class, you will of course receive the revision number of the objects true class.

TYPE

Number

APPLICABILITY

G

30.12 Notify.UserData

NAME

Notify.UserData – set/get user data of object

FUNCTION

A general purpose value to fill in any kind of information. You can get this value later directly from the object. This is a good mechanism to avoid having to use global variables.

The user data you specify here will also be passed to your event handler callback that you have installed using `InstallEventHandler()`. See Section 3.6 [Notifications], page 12, for details.

TYPE

Any

APPLICABILITY

ISG

30.13 Notify.Version

NAME

Notify.Version – get version number of object class

FUNCTION

Get the version number of an objects class. Although `Notify.Version` is documented at notify class, you will of course receive the version number of the objects true class.

TYPE

Number

APPLICABILITY

G

31 Numericbutton class

31.1 Overview

This class is some kind of space-saving slider. It doesn't have any extra attributes, but you can simply use Slider class attributes with this class. See Section 44.1 [Slider class], page 219, for details.

This class requires at least MUI Royale 1.5.

32 Popdrawer class

32.1 Overview

Popdrawer can be used to pop up a standard system asl drawer requester. A separate task is spawned to handle the requesters, the application continues to run.

Popdrawer class will create a string gadget and a popup button for you. You don't need to worry about handling asl requesters. MUI will automatically open one when the popup button is pressed and update the corresponding string gadget when the user terminates the requester. From the programmer's point of view, all you have to do is to handle the string gadgets contents.

32.2 Popdrawer.Acknowledge

NAME

Popdrawer.Acknowledge – get notified when the user hits RETURN

FUNCTION

This attribute will be set to `True` whenever the user hits return in the string gadget. An application can listen to this notification and take the appropriate action.

Using the TAB key or a mouse click to deactivate the gadget will not trigger `Popdrawer.Acknowledge`.

TYPE

Boolean

APPLICABILITY

N

32.3 Popdrawer.Active

NAME

Popdrawer.Active – check if requester is still open

FUNCTION

Popdrawer creates asynchronous popups. Requesters are opened in a separately spawned task and don't disturb the rest of the application. You can ask for the state of a requester by querying the `Popdrawer.Active` attribute. It will return `True` when the requester is currently open, `False` otherwise.

Common use for this attribute is to prevent an application from being terminated while a requester is open. If you try to dispose the Popdrawer object with a currently open requester, MUI will freeze your task as long as the requester stays there.

TYPE

Boolean

APPLICABILITY

G

32.4 Popdrawer.AdvanceOnCR

NAME

Popdrawer.AdvanceOnCR – activate next object in cycle chain (V1.1)

FUNCTION

Set this if you want carriage returns in string gadgets behave like the TAB key, i.e. pressing CR will activate the next/previous gadget in the cycle chain.

TYPE

Boolean

APPLICABILITY

ISG

32.5 Popdrawer.Contents

NAME

Popdrawer.Contents – set/get current popdrawer path

FUNCTION

Get and set the current path of this popdrawer object as shown in the string gadget's contents.

`Popdrawer.Contents` gets updated every time when the contents of the string gadget change. When you set up a notification on this attribute, you will hear about every keystroke.

TYPE

String

APPLICABILITY

ISGN

32.6 Popdrawer.SaveMode

NAME

Popdrawer.SaveMode – enable save mode

FUNCTION

Set this tag to `True` when the file requester is being used for saving. Default is `False`.

TYPE

Boolean

APPLICABILITY

I

32.7 Popdrawer.Title

NAME

Popdrawer.Title – set requester title

FUNCTION

Set the title string for the ASL requester window.

TYPE

String

APPLICABILITY

I

33 Popfile class

33.1 Overview

Popfile can be used to pop up a standard system asl file requester. A separate task is spawned to handle the requesters, the application continues to run.

Popfile class will create a string gadget and a popup button for you. You don't need to worry about handling asl requesters. MUI will automatically open one when the popup button is pressed and update the corresponding string gadget when the user terminates the requester. From the programmer's point of view, all you have to do is to handle the string gadgets contents.

33.2 Popfile.AcceptPattern

NAME
Popfile.AcceptPattern – set accept pattern

FUNCTION
Specifies an AmigaDOS pattern that is used to accept files. That is, only files with names matching this pattern are included in the file list. Default is #? which matches everything.

TYPE
String

APPLICABILITY
I

33.3 Popfile.Acknowledge

NAME
Popfile.Acknowledge – get notified when the user hits RETURN

FUNCTION
This attribute will be set to `True` whenever the user hits return in the string gadget. An application can listen to this notification and take the appropriate action.

Using the TAB key or a mouse click to deactivate the gadget will not trigger `Popfile.Acknowledge`.

TYPE
Boolean

APPLICABILITY
N

33.4 Popfile.Active

NAME

Popfile.Active – check if requester is still open

FUNCTION

Popfile creates asynchronous popups. Requesters are opened in a separately spawned task and don't disturb the rest of the application. You can ask for the state of a requester by querying the `Popfile.Active` attribute. It will return `True` when the requester is currently open, `False` otherwise.

Common use for this attribute is to prevent an application from being terminated while a requester is open. If you try to dispose the popfile object with a currently open requester, MUI will freeze your task as long as the requester stays there.

TYPE

Boolean

APPLICABILITY

G

33.5 Popfile.AdvanceOnCR

NAME

Popfile.AdvanceOnCR – activate next object in cycle chain (V1.1)

FUNCTION

Set this if you want carriage returns in string gadgets behave like the TAB key, i.e. pressing CR will activate the next/previous gadget in the cycle chain.

TYPE

Boolean

APPLICABILITY

ISG

33.6 Popfile.Contents

NAME

Popfile.Contents – set/get current popfile path

FUNCTION

Get and set the current path of this popfile object as shown in the string gadget's contents.

`Popfile.Contents` gets updated every time when the contents of the string gadget change. When you set up a notification on this attribute, you will hear about every keystroke.

TYPE

String

APPLICABILITY
ISGN

33.7 Popfile.Pattern

NAME
Popfile.Pattern – set filter pattern for file requester

FUNCTION
Set the filter pattern for the file requester. Defaults to #?.

TYPE
String

APPLICABILITY
I

33.8 Popfile.RejectIcons

NAME
Popfile.RejectIcons – show/hide icon files

FUNCTION
Set this tag to `True` to cause the requester not to display Workbench icons. Default is `False`.

TYPE
Boolean

APPLICABILITY
I

33.9 Popfile.RejectPattern

NAME
Popfile.RejectPattern – set reject pattern

FUNCTION
Specifies an AmigaDOS pattern that is used to reject files. That is, any files with names matching this pattern are not included in the file list. Default is ~(#?) which matches nothing.

TYPE
String

APPLICABILITY
I

33.10 Popfile.SaveMode

NAME

Popfile.SaveMode – enable save mode

FUNCTION

Set this tag to `True` when the file requester is being used for saving. Default is `False`.

TYPE

Boolean

APPLICABILITY

I

33.11 Popfile.ShowPattern

NAME

Popfile.ShowPattern – show/hide pattern gadget

FUNCTION

Set this tag to `True` to cause a pattern gadget to be displayed. Default is `False`.

TYPE

Boolean

APPLICABILITY

I

33.12 Popfile.Title

NAME

Popfile.Title – set requester title

FUNCTION

Set the title string for the ASL requester window.

TYPE

String

APPLICABILITY

I

34 Popfont class

34.1 Overview

Popfont can be used to pop up a standard system asl font requester. A separate task is spawned to handle the requesters, the application continues to run.

Popfont class will create a string gadget and a popup button for you. You don't need to worry about handling asl requesters. MUI will automatically open one when the popup button is pressed and update the corresponding string gadget when the user terminates the requester. From the programmer's point of view, all you have to do is to handle the string gadgets contents.

Popfont class will display the font name and size separated by a slash in the string gadget, e.g. `topaz/8`.

34.2 Popfont.Acknowledge

NAME

Popfont.Acknowledge – get notified when the user hits RETURN

FUNCTION

This attribute will be set to `True` whenever the user hits return in the string gadget. An application can listen to this notification and take the appropriate action.

Using the TAB key or a mouse click to deactivate the gadget will not trigger `Popfont.Acknowledge`.

TYPE

Boolean

APPLICABILITY

N

34.3 Popfont.Active

NAME

Popfont.Active – check if requester is still open

FUNCTION

Popfont creates asynchronous popups. Requesters are opened in a separately spawned task and don't disturb the rest of the application. You can ask for the state of a requester by querying the `Popfont.Active` attribute. It will return `True` when the requester is currently open, `False` otherwise.

Common use for this attribute is to prevent an application from being terminated while a requester is open. If you try to dispose the Popfont object with a currently open requester, MUI will freeze your task as long as the requester stays there.

TYPE

Boolean

APPLICABILITY

G

34.4 Popfont.AdvanceOnCR

NAME

Popfont.AdvanceOnCR – activate next object in cycle chain (V1.1)

FUNCTION

Set this if you want carriage returns in string gadgets behave like the TAB key, i.e. pressing CR will activate the next/previous gadget in the cycle chain.

TYPE

Boolean

APPLICABILITY

ISG

34.5 Popfont.Contents

NAME

Popfont.Contents – set/get string gadget contents

FUNCTION

Get and set a string gadget's contents. Popfont class uses a slash to separate font name and size, e.g. `topaz/8`.

`Popfont.Contents` gets updated every time when the contents of the string gadget change. When you set up a notification on this attribute, you will hear about every keystroke.

TYPE

String

APPLICABILITY

ISGN

34.6 Popfont.FixedWidthOnly

NAME

Popfont.FixedWidthOnly – show only fixed width fonts

FUNCTION

Set this tag to `True` to cause the requester to only display fixed-width fonts. Default is `False`.

TYPE

Boolean

APPLICABILITY
I

34.7 Popfont.MaxHeight

NAME
Popfont.MaxHeight – set maximum font height

FUNCTION
The maximum font height to let the user select. Default is 24.

TYPE
Number

APPLICABILITY
I

34.8 Popfont.MinHeight

NAME
Popfont.MinHeight – set minimum font height

FUNCTION
The minimum font height to let the user select. Default is 5.

TYPE
Number

APPLICABILITY
I

34.9 Popfont.Title

NAME
Popfont.Title – set requester title

FUNCTION
Set the title string for the ASL requester window.

TYPE
String

APPLICABILITY
I

35 Poplist class

35.1 Overview

Poplist class simplifies creation of popups that contain just a simple list of predefined gadget contents. Poplist class will create a string gadget and a popup button for you. Whenever the user hits the popup button, a window containing a list of predefined entries will be opened so that the user can choose one from the list.

When creating a poplist object, you have to use the `<item>` tag to fill it with entries. Here is an example:

```
<poplist>
   <item>The</item>
   <item>quick</item>
   <item>brown</item>
   <item>fox</item>
   <item>jumps</item>
   <item>over</item>
   <item>the</item>
   <item>lazy</item>
   <item>dog</item>
</poplist>
```

The list entries that you specify using the `<item>` tag can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

35.2 Poplist.Acknowledge

NAME

Poplist.Acknowledge – get notified when the user hits RETURN

FUNCTION

This attribute will be set to `True` whenever the user hits return in the string gadget. An application can listen to this notification and take the appropriate action.

Using the TAB key or a mouse click to deactivate the gadget will not trigger `Poplist.Acknowledge`.

TYPE

Boolean

APPLICABILITY

N

35.3 Poplist.AdvanceOnCR

NAME

Poplist.AdvanceOnCR – activate next object in cycle chain (V1.1)

FUNCTION

Set this if you want carriage returns in string gadgets behave like the TAB key, i.e. pressing CR will activate the next/previous gadget in the cycle chain.

TYPE

Boolean

APPLICABILITY

ISG

35.4 Poplist.Contents

NAME

Poplist.Contents – set/get current poplist contents

FUNCTION

Get and set the current contents of this poplist object as shown in the string gadget's contents.

`Poplist.Contents` gets updated every time when the contents of the string gadget change. When you set up a notification on this attribute, you will hear about every keystroke.

TYPE

String

APPLICABILITY

ISGN

36 Poppen class

36.1 Overview

Poppen class adds input capabilities to its super class Pendisplay. It should be used if your application allows users to configure some custom pens for rendering.

A Poppen object will appear as kind of a button which displays the currently selected color. When the user hits the button, a Popup window containing a Penadjust object opens up and lets the user choose change the color.

You can control the window title of the popup window using the `Poppen.Title` attribute on the Poppen object. It will remember its value and use it when creating the popup window.

As most MUI popups, the Penadjust popup window runs asynchronously and stays there until the user terminates it with `"OK"` or `"Cancel"`. Furthermore, if the popup window is automatically cancelled if the pop button's root window gets closed.

You can get/set the current color from a Poppen object by using the `Poppen.RGB` attribute.

You can find some example code on using this class in the Class2 demo of the MUI distribution.

36.2 Poppen.RGB

NAME

Poppen.RGB – set/get color

FUNCTION

Set or get the poppen color. If you set up a notification on this attribute, you will be notified whenever the color changes. From the Hollywood script the color is specified as a simple numerical value containing 8 bits for each component. When you specify the color in the XML file, it has to be passed as a 6 character string prefixed by the #-character (just like in HTML).

TYPE

Number

APPLICABILITY

ISGN

36.3 Poppen.Title

NAME

Poppen.Title – set/get popup window title

FUNCTION

Set or get the window title of the popup window.

TYPE

String

APPLICABILITY

ISG

37 Prop class

37.1 Overview

Prop class generates the well known proportional gadgets. It offers the same attributes as a usual boopsi gadget of propgclass. However, MUI's prop gadgets allow using any imagery for the knob and for the background.

Note that this class won't create any navigation buttons for the scrollbar. If you'd like to have two navigation buttons to be connected with the scrollbar, use Scrollbar class instead. See Section 42.1 [Scrollbar class], page 215, for details.

37.2 Prop.Decrease

NAME

Prop.Decrease – decrease value of prop gadget

SYNOPSIS

```
mui.DoMethod(id, "Decrease", amount)
```

FUNCTION

This method decreases the value of a proportional gadget by the specified amount. Negative values are ok. Range checking is done automatically.

INPUTS

`id`	id of the application object
`amount`	amount to substract from the gadget's current position.

37.3 Prop.Entries

NAME

Prop.Entries – set/get number of entries

FUNCTION

Set or get the total number of entries.

TYPE

Number

APPLICABILITY

ISG

37.4 Prop.First

NAME

Prop.First – set/get number of first entry

FUNCTION

Set or get the number of the first entry.

TYPE

Number

APPLICABILITY

ISGN

37.5 Prop.Horiz

NAME

Prop.Horiz – set prop object direction

FUNCTION

Determine if you want a horizontal or a vertical prop gadget.

Defaults to `False`, i.e. vertical.

TYPE

Boolean

APPLICABILITY

IG

37.6 Prop.Increase

NAME

Prop.Increase – increase value of prop gadget

SYNOPSIS

```
mui.DoMethod(id, "Increase", amount)
```

FUNCTION

This method increases the value of a proportional gadget by the specified amount. Negative values are ok. Range checking is done automatically.

INPUTS

`id`	id of the application object
`amount`	amount to add to the gadget's current position.

37.7 Prop.Slider

NAME

Prop.Slider – put prop into slider mode

FUNCTION

Indicate that this prop gadget is used in a slider. MUI might then use different imagery. Since you really should use the slider class when creating sliders, you normally don't need to care about this attribute.

TYPE

Boolean

APPLICABILITY

ISG

37.8 Prop.UseWinBorder

NAME

Prop.UseWinBorder – put prop object into window border

FUNCTION

If you set this attribute some very special magic will take place with this proportional gadget. In fact, it will not eat any display space at all or render anything, it stays invisible in your windows GUI. Instead, the gadgets control and display will be linked to one of the scrollbars in the window border.

There is no difference in talking to the object, you can use all prop gadget attributes regardless whether its a real prop gadget or just a window border link. Of course, gadgets in window borders will use the intuition look regardless what the user has configured.

You *must* enable border scrollers for the parent window with the corresponding attributes before using `Prop.UseWinBorder`.

Obviously, you can only link exactly one prop gadget to one window scroller. Linking more than one gadget to the same window scroller will result in big confusion.

The following values are recognized by this attribute:

`Left`	Use left window border.
`Right`	Use right window border.
`Bottom`	Use bottom window border.

TYPE

String (see above for possible values)

APPLICABILITY

I

37.9 Prop.Visible

NAME

Prop.Visible – set/get number of visible entries

FUNCTION

Set or get the number of visible entries.

TYPE

Number

APPLICABILITY

ISG

38 Radio class

38.1 Overview

Radio class generates radio button gadgets. They do the same job as cycle gadgets and eat up more window space, maybe that's the reason why so few of them can be found in existing applications.

When you declare a radio gadget, you have to use the `<item>` tag to fill the radio gadget with items. Every radio gadget needs to have at least one item.

Here is an example XML excerpt for creating a radio gadget:

```
<radio id="printer">
    <item>HP Deskjet</item>
    <item>NEC P6</item>
    <item>Okimate 20</item>
</radio>
```

The radio entries that you specify using the `<item>` tag can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

38.2 Radio.Active

NAME

Radio.Active – set/get active radio item

FUNCTION

This attributes defines the number of the active entry in the radio gadgets. Valid range is from 0 for the first entry to NumEntries-1 for the last.

Setting `Radio.Active` causes the gadget to be updated. On the other hand, when the user plays around with the gadget, `Radio.Active` will always reflects the current state.

TYPE

Number

APPLICABILITY

ISGN

39 Rectangle class

39.1 Overview

Rectangle class seems kind of useless since it does not define many attributes or methods itself. However, objects of this type are frequently used in every application. They allow insertion of space to control MUI's layout process.

You can use the attributes of the super class to control the size and other attributes of the rectangle. E.g. you can use `Area.FixWidth` and `Area.FixHeight` to create rectangles of a fixed size.

Rectangle objects are also often necessary as padding space next to objects which are not resizable themselves. The `<label>` object is e.g. not resizable. To prevent these static label objects from blocking the resizing feature of your whole GUI, you can simply pad them with a `<rectangle>` object. Here is an example of a label next to a checkmark object, padded using an invisible rectangle:

```
<hgroup>
   <label>Enable hardware acceleration</label>
   <checkmark/>
   <rectangle/>
</hgroup>
```

39.2 Rectangle.BarTitle

NAME
Rectangle.BarTitle – define title text for horizontal bar

FUNCTION
This attribute describes a text string which will be displayed in group title style centered in the rectangle. Really only makes sense if `Rectangle.HBar` is set to `True`.

TYPE
String

APPLICABILITY
IG

39.3 Rectangle.HBar

NAME
Rectangle.HBar – add horizontal bar to rectangle

FUNCTION
When set to `True`, MUI draws a horizontal bar in the middle of the rectangle. Such bars can be used instead of group frames to separate objects in a window.

TYPE
Boolean

APPLICABILITY

IG

39.4 Rectangle.VBar

NAME

Rectangle.VBar – add vertical bar to rectangle

FUNCTION

When set to `True`, MUI draws a vertical bar in the middle of the rectangle. Such bars can be used instead of group frames to separate objects in a window.

TYPE

Boolean

APPLICABILITY

IG

40 Register class

40.1 Overview

Register class is a special class for handling multi page groups. Using this class, you only have to supply an array of groups, describing the register's children. How these children are visualized, either with a cycle gadget of with a register-like group, is the choice of the user. You need to use the `Group.Title` attribute to give your register tabs titles.

Here is an example of a three page register group:

```
<register>
   <vgroup title="Page 1">
       <listview>
          <column>
             <item>Entry</item>
          </column>
       </listview>
   </vgroup>
   <vgroup title="Page 2">
       <texteditor/>
   </vgroup>
   <vgroup title="Page 3">
       <button>Click me</button>
   </vgroup>
</register>
```

40.2 Register.ActivePage

NAME

Register.ActivePage – set/get active page of register group

FUNCTION

Set (or get) the active page of a register group. Only this active page is displayed, all others are hidden.

The value may range from 0 (for the first child) to numchildren-1 (for the last child). Children are adressed in the order of creation.

The following special values are possible when setting the active page:

`First`	First page.
`Last`	Last page.
`Prev`	Previous page.
`Next`	Next page.
`Advance`	Advance page.

Note: You may *never* supply an incorrect page value!

TYPE

Number or string (see above for possible values)

APPLICABILITY

ISGN

40.3 Register.AddPage

NAME

Register.AddPage – add new page to register (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "AddPage", obj)
```

FUNCTION

This method can be used to add a new page to a register group. The new page specified by `"obj"` must be a detached group object. This detached group object will then be added as the register's last page. After this method returns the specified group object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Register.ClosePage` method.

This feature requires at least MUI 4.0.

INPUTS

`id`	id of the register object
`obj`	id of the group object to add as a new page

40.4 Register.Closable

NAME

Register.Closable – add close gadget to register pages (V1.2)

FUNCTION

If you set this attribute to `True`, MUI will add close gadgets to each individual tab page. When the user hits the close gadget, `Register.CloseRequest` will trigger.

This feature requires at least MUI 4.0.

TYPE

Boolean

APPLICABILITY

I

40.5 Register.ClosePage

NAME

Register.ClosePage – remove page from register group (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "ClosePage", obj)
```

FUNCTION

This method will close the page tab specified by "obj". The group object that constitutes the page that is being closed will then change its state from attached to detached. This means that you could attach it now to another register or group using a function like `Group.Insert` and `Register.AddPage`, respectively, or you could free it using `mui.FreeObject()`.

This feature requires at least MUI 4.0.

INPUTS

`id`	id of the register object
`obj`	id of the page to remove

40.6 Register.CloseRequest

NAME

Register.CloseRequest – handle close request of page (V1.2)

FUNCTION

When the user hits a register page's close gadget, the page isn't closed immediately. Instead MUI only sets this attribute to `True` to allow your application to react.

Usually, you will setup a notification that automatically closes the page when a close request appears, but you could e.g. pop up a confirmation requester or do some other things first.

This feature requires at least MUI 4.0.

TYPE

Boolean

APPLICABILITY

N

40.7 Register.GetPageID

NAME

Register.GetPageID – return ID of a page inside the register group (V1.2)

SYNOPSIS

```
id$ = mui.DoMethod(id, "GetPageID", idx)
```

FUNCTION

This method returns the ID of the page at the specified index within the register group. "idx" can be an absolute number ranging from 0 to the number of pages in the register minus 1 or it can be one of the following special values:

`First`	First page.
`Last`	Last page.
`Active`	Active page.

This feature requires at least MUI 4.0.

INPUTS

`id`	id of the register object
`idx`	absolute index of desired page or special string constant (see above)

RESULTS

`id$`	id of the page object at the specified register index

40.8 Register.InsertPage

NAME

Register.InsertPage – insert new page into register (V1.4)

SYNOPSIS

```
mui.DoMethod(id, "InsertPage", obj, pos)
```

FUNCTION

This method can be used to insert a new page into a register group. The new page specified by "obj" must be a detached group object. This detached group object will then be inserted into the register position specified by "pos". This is an integer position starting from 0 for the first page. After this method returns the specified group object will change its state from detached to attached. That is why you must no longer use functions that expect a detached object with this object now.

Detached MUI objects can be created either by calling the `mui.CreateObject()` function or by explicitly detaching them from their parent by using the `Register.ClosePage` method.

This feature requires at least MUI 4.0.

INPUTS

`id`	id of the register object
`obj`	id of the group object to insert as a new page
`pos`	insertion position starting from 0 for the first page

40.9 Register.Pages

NAME

Register.Pages – find out number of pages in register (V1.2)

FUNCTION

You can query this attribute to find out the current number of pages in the register object.

This feature requires at least MUI 4.0.

TYPE

Number

APPLICABILITY

G

40.10 Register.Position

NAME

Register.Position – set position of page tabs (V1.2)

FUNCTION

This attribute allows you to configure whether the page tab browser should appear to the top, left, right, or bottom of the pages.

The following values are recognized by this attribute:

`Left`	Put tab browser to the left of the pages.
`Right`	Put tab browser to the right of the pages.
`Bottom`	Put tab browser to the bottom of the pages.
`Top`	Put tab browser to the top of the pages.

This feature requires at least MUI 4.0.

TYPE

String (see above for possible values)

APPLICABILITY

I

41 Scale class

41.1 Overview

A Scale object generates a percentage scale running from 0% to 100%. A good place for such an object is e.g. below a fuel gauge.

Depending on how much space is available, the scale will be more or less detailed.

Due to MUI's automatic layout system, you don't need to worry about it's size. When placed in a vertical group just below the object you want to scale, everything is fine.

Scale class is often used together with Gauge class. See Section 16.1 [Gauge class], page 97, for details.

41.2 Scale.Horiz

NAME

Scale.Horiz – set/get scale alignment

FUNCTION

Indicate whether you want a horizontal or a vertical scale.

Currently, only the horizontal scale is implemented.

TYPE

Boolean

APPLICABILITY

ISG

42 Scrollbar class

42.1 Overview

Scrollbar class creates a proportional gadget and two button gadgets with approriate imagery to make up a scrollbar. Since Scrollbar class is a subclass of Prop class, you can talk and listen to a scrollbar as if it was just a single prop gadget. See Section 37.1 [Prop class], page 199, for details.

This class requires at least MUI Royale 1.4.

42.2 Scrollbar.IncDecSize

NAME

Scrollbar.IncDecSize – set/get scrolling amount on button click (V1.4)

FUNCTION

Set the amount by which the scrollbar position is increased or decreased whenever one of the arrow buttons is clicked. Defaults to 1.

TYPE

Number

APPLICABILITY

ISG

42.3 Scrollbar.Type

NAME

Scrollbar.Type – set scrollbar type (V1.4)

FUNCTION

Specify a certain scrollbar type. Normally, you should respect the users choice and avoid using this attribute.

The following inputs are accepted:

`Default`	Default type.
`Bottom`	Bottom type.
`Top`	Top type.
`Sym`	Sym type.
`None`	No type.

TYPE

String (see above for possible values)

APPLICABILITY

I

43 Scrollgroup class

43.1 Overview

Scrollgroup objects can be used to supply virtual groups with scrollbars. These scrollbars automatically adjust according to the virtual and display sizes of the underlying virtual group. When scrolling is unnecessary (i.e. the virtual group is completely visible), the scrollers might get disabled or even disappear completely, depending on the users preferences settings.

It is important to note that scrollgroup objects do not work with normal groups. Instead, you need to give them a virtual group as a child, i.e. you need to use the `<virtgroup>` tag to create a MUI object of type Virtgroup class. See Section 50.1 [Virtgroup class], page 277, for details.

Here is an XML example:

```
<scrollgroup>
   <virtgroup>
      <radio>
         <item>Amiga 500</item>
         <item>Amiga 1200</item>
         <item>Amiga 4000</item>
      </radio>
      <listview>
         <column/>
      </listview>
   </virtgroup>
</scrollgroup>
```

The XML code above embeds a radio and a listview object inside a virtual group which in turn is embedded inside a scrollgroup.

43.2 Scrollgroup.FreeHoriz

NAME

Scrollgroup.FreeHoriz – allow horizontal scrolling

FUNCTION

Specify if a scroll group should be horizontally moveable. Defaults to `True`.

TYPE

Boolean

APPLICABILITY

I

43.3 Scrollgroup.FreeVert

NAME

Scrollgroup.FreeVert – allow vertical scrolling

FUNCTION

Specify if a scroll group should be vertically moveable. Defaults to `True`.

TYPE

Boolean

APPLICABILITY

I

43.4 Scrollgroup.UseWinBorder

NAME

Scrollgroup.UseWinBorder – use window scrollbars for this scroll group

FUNCTION

If you set this to `True`, MUI will automatically make this scrollgroup controllable from gadgets in the window border. MUI will use the right border scroller if the virtual groups is allowed to move vertically and the bottom border scroller if the virtual group is allowed to move horizontally.

You must set the corresponding window attributes, e.g. you have to set the attribute `Window.UseRightBorderScroller` for your parent window to use this feature.

TYPE

Boolean

APPLICABILITY

I

44 Slider class

44.1 Overview

The slider class generates a GUI element that allows a user to adjust a numeric value. The programmer doesn't have very much influence on the slider's outfit, there are only very few tags available. Future versions of MUI will probably include some preferences options to allow the user (*not* the programmer) to configure this outfit.

Note that since slider is a subclass of group class, you can get horizontal or vertical sliders by simply using the `Slider.Horiz` attribute. Default is a horizontal slider.

Here is an XML example of a slider that allows the user to configure a number ranging from 0 to 100:

```
<slider min="0" max="100"/>
```

44.2 Slider.Format

NAME

Slider.Format – customize slider text

FUNCTION

A template string to describe the format of the slider display. Whenever Slider.mui thinks it's time to render a new value, it doesn't simply write it to a string but uses the format mask specified in this attribute to compose the new string that is then rendered to the slider display. Any "%ld" in the format string is replaced by the current slider level.

Note well: The maximum length of the result string for `Slider.Format` is limited to 32 characters. If you need more, you must use `Slider.Stringify`.

`Slider.Format` defaults to "%ld".

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

ISG

EXAMPLE

```
mui.Set(obj, "Format", "Custom text: %ld")
```

The code above sets the slider text to "Custom text" followed by the current slider level.

44.3 Slider.Horiz

NAME

Slider.Horiz – set/get alignment of slider

FUNCTION

Specify if you want a horizontal or vertical slider.

TYPE

Boolean

APPLICABILITY

ISG

44.4 Slider.Level

NAME

Slider.Level – set/get current slider level

FUNCTION

The current position of the slider knob. This value is guaranteed to be between `Slider.Min` and `Slider.Max`.

TYPE

Number

APPLICABILITY

ISGN

44.5 Slider.Max

NAME

Slider.Max – set/get maximum level of slider

FUNCTION

Adjust the maximum value for a slider object.

TYPE

Number

APPLICABILITY

ISG

44.6 Slider.Min

NAME

Slider.Min – set/get minimum level of slider

FUNCTION

Adjust the minimum value for a slider object. Of course you can use negative number, e.g. for a slider to adjust task priority.

TYPE

Number

APPLICABILITY
ISG

44.7 Slider.Pressed

NAME
Slider.Pressed – learn when slider knob is pressed (V1.5)

FUNCTION
This attribute is triggered when the user presses the slider knob.

TYPE
Boolean

APPLICABILITY
N

44.8 Slider.Quiet

NAME
Slider.Quiet – hide current level from slider

FUNCTION
When set to `True`, the slider doesn't display it's current level in a text object.

TYPE
Boolean

APPLICABILITY
I

44.9 Slider.Reverse

NAME
Slider.Reverse – reverse direction of slider

FUNCTION
Setting this attribute to `True` will reverse the direction of the slider.

TYPE
Boolean

APPLICABILITY
ISG

44.10 Slider.Stringify

NAME

Slider.Stringify – setup callback for custom slider text

FUNCTION

This attribute can be used to set up a callback function that transforms a slider level into a custom slider text. The callback function you specify here will be called like a standard MUI Royale event callback, but with the following extra argument:

`Value:` Contains the current slider level.

Your callback function then has to return a string that should be displayed in the slider gadget. You might e.g. want to display a nice formatted time string (hh:mm:ss) in a slider which adjusts a number of seconds. Or you need to adjust a baudrate from a hand of predefined values. Just use `Slider.Stringify` and you have the choice how the slider value translates into a string.

Note that you always have to pass a string specifying the name of a Hollywood function to this attribute. Never pass the function directly but always pass the name of the function as a string!

Also note that you must also set up a notification on this attribute. Otherwise the callback function will never get called.

See Section 3.6 [Notifications], page 12, for details.

TYPE

String

APPLICABILITY

ISGN

EXAMPLE

```
<slider notify="stringify" stringify="p_Callback" min="0" max="64"
   level="32" format="--Full--"/>

Function p_Callback(msg)
  If msg.value = 0 Then Return("Mute")
  If msg.value = 64 Then Return("Full")
  Return(msg.value)
EndFunction
```

The code above creates a slider for controlling sound volume. If the slider is all the way to the left, "Mute" will be displayed. If it is all the way to the right, "Full" will be displayed. Note that we use the "Format" tag in the XML file to specify an initial width for the slider knob so that it is large enough for all strings that are going to be returned by our callback.

45 String class

45.1 Overview

String class generates standard string gadgets with all editing facilities (clear, undo, etc.) enabled. By default, string gadgets do not have any label next to them. If you want to have a label next to your string gadget, you need to put it into a `<hgroup>` and then use Label class or Text class class to put a label next to it.

Here is an XML example of a string gadget:

```
<string id="mystring" contents="Enter something!"/>
```

45.2 String.Accept

NAME

String.Accept – set/get characters accepted by string gadget

FUNCTION

A string containing characters allowed as input for the string gadget. Whenever the user hits a character not found in `String.Accept`, he will hear a beep and gadgets contents won't have changed.

TYPE

String

APPLICABILITY

ISG

EXAMPLE

```
mui.Set(obj, "Accept", "0123456789-")
```

The above code will set a string gadget to only accept numbers and hyphens.

45.3 String.Acknowledge

NAME

String.Acknowledge – get notified when the user hits RETURN

FUNCTION

This attribute will be set to `True` whenever the user hits return in the gadget. An application can listen to this notification and take the appropriate action.

Using the TAB key or a mouse click to deactivate the gadget will not trigger `String.Acknowledge`.

TYPE

Boolean

APPLICABILITY

N

45.4 String.AdvanceOnCR

NAME

String.AdvanceOnCR – activate next object in cycle chain

FUNCTION

Set this if you want carriage returns in string gadgets behave like the TAB key, i.e. pressing CR will activate the next/previous gadget in the cycle chain.

TYPE

Boolean

APPLICABILITY

ISG

45.5 String.Contents

NAME

String.Contents – set/get string gadget contents

FUNCTION

Get and set a string gadgets contents.

`String.Contents` gets updated every time when the contents of the string gadget change. When you set up a notification on this attribute, you will hear about every keystroke.

If you try to set contents to something larger than `String.MaxLen` MUI will silently strip the additional characters.

TYPE

String

APPLICABILITY

ISGN

45.6 String.Copy

NAME

String.Copy – copy marked text (V1.3)

SYNOPSIS

```
mui.DoMethod(id, "Copy")
```

FUNCTION

Copy currently selected text to clipboard.

This method requires MUI 4.0 or better.

INPUTS

`id`	id of the string object

45.7 String.CursorPos

NAME

String.CursorPos – set/get cursor position (V1.3)

FUNCTION

Sets the current cursor position.

This attributes requires MUI 4.0 or better.

TYPE

Number

APPLICABILITY

SG

45.8 String.Cut

NAME

String.Cut – cut marked text (V1.3)

SYNOPSIS

```
mui.DoMethod(id, "Cut")
```

FUNCTION

Cut currently selected text and put it in the clipboard.

This method requires MUI 4.0 or better.

INPUTS

`id`	id of the string object

45.9 String.Insert

NAME

String.Insert – insert text (V1.3)

SYNOPSIS

```
mui.DoMethod(id, "Insert", t$)
```

FUNCTION

This will insert the given text t$ at the current cursor position.

This method requires MUI 4.0 or better.

INPUTS

`id`	id of the string object
`t$`	text to insert

45.10 String.MarkEnd

NAME

String.MarkEnd – set/get end position of marked text (V1.3)

FUNCTION

Sets the end position of marked text. Set this to -1 to remove any text marking.

This attributes requires MUI 4.0 or better.

TYPE

Number

APPLICABILITY

ISG

45.11 String.MarkStart

NAME

String.MarkStart – set/get start position of marked text (V1.3)

FUNCTION

Sets the start position of marked text. Set this to -1 to remove any text marking.

This attributes requires MUI 4.0 or better.

TYPE

Number

APPLICABILITY

ISG

45.12 String.MaxLen

NAME

String.MaxLen – set/get maximum length of string

FUNCTION

Setup the maximum length for the string gadget. This attribute is only valid at object creation time.

Default maximum length is 80.

NOTE: The maximum length includes the 0-byte at the end of the string. To let the user enter e.g. 10 characters, you would have to specify a maxlen of 11.

TYPE

Number

APPLICABILITY

IG

45.13 String.Paste

NAME

String.Paste – paste text from clipboard (V1.3)

SYNOPSIS

```
mui.DoMethod(id, "Paste")
```

FUNCTION

Pastes text from clipboard into the string gadget.

This method requires MUI 4.0 or better.

INPUTS

`id`	id of the string object

45.14 String.Redo

NAME

String.Redo – redo last operation (V1.3)

SYNOPSIS

```
mui.DoMethod(id, "Redo")
```

FUNCTION

Redo last operation of string gadget.

This method requires MUI 4.0 or better.

INPUTS

`id`	id of the string object

45.15 String.Reject

NAME

String.Reject – set/get characters rejected by string gadget

FUNCTION

A string containing characters that should not be accepted as input for the string gadget. Whenever the user hits such a char, he will hear a beep and gadgets contents won't have changed.

TYPE

String

APPLICABILITY

ISG

EXAMPLE

```
mui.Set(obj, "Reject", "0123456789")
```

The above code will set a string gadget to reject numbers.

45.16 String.Secret

NAME

String.Secret – hide user input

FUNCTION

This attribute causes the string gadget to display only dots instead of the real contents. Useful for password requesters.

TYPE

Boolean

APPLICABILITY

IG

45.17 String.Undo

NAME

String.Undo – undo last operation (V1.3)

SYNOPSIS

```
mui.DoMethod(id, "Undo")
```

FUNCTION

Undo last operation of string gadget.

This method requires MUI 4.0 or better.

INPUTS

`id`	id of the string object

46 Text class

46.1 Overview

Text class allows generating objects that contain some kind of text. You can control the outfit of your text with some special control characters, including italics, bold, underline and color codes. Format codes align text either left, centered or right, linefeeds allow multiline text fields.

Please note that text class does not offer automatic word wrapping. If you want to have on-the-fly word wrapping, you have to use floattext class instead. See Section 15.1 [Floattext], page 95, for details.

Here is an example of how to use the `<text>` command:

```
<text>Hello World</text>
```

Here is the same example in bold:

```
<text>\33bHello World</text>
```

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

46.2 Text.Contents

NAME

Text.Contents – set/get text object contents

FUNCTION

String to be displayed in a text object.

If the string is larger than available display space, it will be clipped. Setting `Text.Contents` to `""` results in an empty text object.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

SG

46.3 Text.HiChar

NAME

Text.HiChar – highlight single character

FUNCTION

If the character given here exists in the displayed string (no matter if upper or lower case), it will be underlined. This is useful in connection with `Area.ControlChar`.

TYPE

Single character string

APPLICABILITY

I

46.4 Text.PreParse

NAME

Text.PreParse – set/get preparse string for text object

FUNCTION

String containing format definitions to be parsed before the text from `Text.Contents` is printed.

Using this tag, you can easily define different formats, colors and styles without modifying the original string. For example, if you set `Text.PreParse` to `"\33c"`, the text in the text object will always be centered.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

ISG

46.5 Text.SetMax

NAME

Text.SetMax – limit maximum width

FUNCTION

Boolean value to indicate whether the object's maximal width shall be calculated to fit the string given with `Text.Contents`.

When set to `False`, maximum width is not limited.

For a text object that needs to be updated (e.g. some information about your programs status) you would probably set `Text.SetMax` to `False` to allow resizing of this object.

For a label for one of your gadgets, you might want to give this tag a value of `True` to prevent MUI from inserting additional layout space.

Defaults to `False`.

TYPE

Boolean

APPLICABILITY

I

46.6 Text.SetMin

NAME

Text.SetMin – set a minimum width for text object

FUNCTION

Boolean value to indicate wether the objects minimal width shall be calculated to fit the string given with `Text.Contents`.

When set to `False`, minimum width will be set to 0 and the displayed string may be clipped.

Defaults to `True`.

TYPE

Boolean

APPLICABILITY

I

46.7 Text.SetVMax

NAME

Text.SetVMax – limit object height

FUNCTION

Settings this to `False` makes a TextObjects y-size unlimited. Defaults to `True` which means the objects height is fixed.

TYPE

Boolean

APPLICABILITY

I

47 Texteditor class

47.1 Overview

`TextEditor.mcc` is a multiline string gadget which holds most of the functions of a normal text editor including some special functionality for importing/exporting text from/to special formats.

The gadget was originally written in 1997 by Allan Odgaard. As of version 15.10, released in April 2005, the gadget is maintained by the `TextEditor.mcc` Open Source team.

It is released and distributed under the terms of the GNU Lesser General Public License (LGPL) and available free of charge.

Please visit `http://www.sf.net/projects/texteditor-mcc/` for the very latest version and information regarding `TextEditor.mcc`.

Here is an XML example of how to include a text editor object in your GUI:

```
<texteditor scrollbar="true" contents="Enter your text here!"/>
```

47.2 Texteditor.ActiveObjectOnClick

NAME

Texteditor.ActiveObjectOnClick – automatically become active object

FUNCTION

This tag allows to set/get whether the texteditor object is automatically set the `Window.ActiveObject` object of the window in case the user performs a mouse action (click/select) in the texteditor. As a consequence, the texteditor will then get the full keyboard focus and will eventually signal its active status by highlighting a marked area with the active color rather than using the inactive marked color in inactive state.

Please note that depending on the `Texteditor.ReadOnly` setting, this attribute has different default values. In case the texteditor is set to `Texteditor.ReadOnly` while initialized this attribute will be set to `False` as a default. In contrast, when the object is in full write mode it will be set to `True` instead. However, during operation this behaviour can be overridden by setting this attribute to another value.

TYPE

Boolean

APPLICABILITY

ISG

47.3 Texteditor.Align

NAME

Texteditor.Align – set/get current paragraph's alignment

FUNCTION

Set/get the current paragraph's alignment.

If an area is marked while you set this attribute, then the new alignment will be set for the complete area.

The following values are possible:

`Left`	Left alignment.
`Right`	Right alignment.
`Center`	Centered alignment.
`Justified`	Justified alignment (not implemented yet).

TYPE
String (see above for possible values)

APPLICABILITY
SGN

47.4 Texteditor.AreaMarked

NAME
Texteditor.AreaMarked – learn about marked areas

FUNCTION
This tag will be set to `True` when text is marked, and back to `False` when nothing is marked. You can create a notify event with this tag and let your cut/copy buttons become ghosted when nothing is marked.

TYPE
Boolean

APPLICABILITY
GN

47.5 Texteditor.AutoClip

NAME
Texteditor.AutoClip – enable/disable auto clipboard copy

FUNCTION
When the gadget is in read only mode and the user marks some text, then it will be automatically copied to the clipboard. With this tag you can disable that behaviour, but think twice, because the configured 'copy' key (which the user normally uses to copy text) will only function when the gadget is active (or default) which it won't automatically become in read only mode (when clicked).

TYPE
Boolean

APPLICABILITY
ISG

47.6 Texteditor.Clear

NAME

Texteditor.Clear – clear text

SYNOPSIS

`mui.DoMethod(id, "Clear")`

FUNCTION

This will clear all the text in the gadget.

INPUTS

`id`	id of the text editor object

47.7 Texteditor.Color

NAME

Texteditor.Color – set/get current text color (V1.2)

FUNCTION

This attribute can be used to set/get the current text color. Any text entered after setting this attribute will appear in the specified color. To change the color of existing text, use the `Texteditor.SetColor` method instead.

You can also setup a notification on this attribute to learn when the cursor has been moved over text in a different color.

From the Hollywood script the color is specified as a simple numerical value containing 8 bits for each component. When you specify the color in the XML file, it has to be passed as a 6 character string prefixed by the #-character (just like in HTML).

TYPE

Number

APPLICABILITY

SGN

47.8 Texteditor.ColorMap

NAME

Texteditor.ColorMap – get text editor's current color map (V1.2)

FUNCTION

This attribute can be used to get the editor's current color map. As Texteditor class is still palette-based, this will always return a table containing 256 entries that describe which colors the text editor's pens map to. This is needed if you want to identify colors used in a text returned by `Texteditor.Contents`. If a text uses different colors, you will only get the information about the different pens used by the text. You can then use this table to map the pen information to RGB colors.

TYPE

Table

APPLICABILITY
G

47.9 Texteditor.Columns

NAME
Texteditor.Columns – set desired gadget width

FUNCTION
Set the desired width, in characters.

TYPE
Number

APPLICABILITY
I

47.10 Texteditor.Contents

NAME
Texteditor.Contents – set/get text editor contents

FUNCTION
Use this attribute to set or get the contents of the texteditor object.

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE
String

APPLICABILITY
ISG

47.11 Texteditor.ConvertTabs

NAME
Texteditor.ConvertTabs – convert tabs to spaces

FUNCTION
When `True` (default) TextEditor will convert tabs (\t) to the number of spaces specified by the user configuration setting. If `False`, TextEditor will instead put pure \t characters and just display spaces to the user.

Please note that a change of this attribute from `True` to `False` will cause an update of editor gadget (but cursor coordinates will be reset to zeroes).

TYPE
Boolean

APPLICABILITY

ISG

47.12 Texteditor.Copy

NAME

Texteditor.Copy – copy marked text

SYNOPSIS

```
mui.DoMethod(id, "Copy")
```

FUNCTION

Copy currently selected text to clipboard.

INPUTS

`id`	id of the text editor object

47.13 Texteditor.CursorX

NAME

Texteditor.CursorX – set/get cursor x position

FUNCTION

You can get or set the cursor's X position with this tag. The first character on a line has position 0. The position is not affected by the gadget's autowrap feature. If you set a value higher than the length of the current line, then it will be automatically truncated.

TYPE

Number

APPLICABILITY

ISGN

47.14 Texteditor.CursorY

NAME

Texteditor.CursorY – set/get cursor y position

FUNCTION

You can get or set the cursor's Y position with this tag. The first line has position 0. The position is not affected by the gadget's autowrap feature. If you set a value higher than the number of lines, then it will be automatically truncated.

TYPE

Number

APPLICABILITY

ISGN

47.15 Texteditor.Cut

NAME

Texteditor.Cut – cut marked text

SYNOPSIS

```
mui.DoMethod(id, "Cut")
```

FUNCTION

Cut currently selected text and put it in the clipboard.

INPUTS

`id`	id of the text editor object

47.16 Texteditor.Erase

NAME

Texteditor.Erase – erase all text

SYNOPSIS

```
mui.DoMethod(id, "Erase")
```

FUNCTION

Erases all text in the text editor gadget.

INPUTS

`id`	id of the text editor object

47.17 Texteditor.ExportHook

NAME

Texteditor.ExportHook – set text editor's export hook (V1.4)

FUNCTION

Depending on the inputs a different export hook will be fired as soon as text is exported from the text editor gadget. The default is to export the currently shown data as is. That means, that all text, including the escape sequences for showing soft styles will be exported.

You can change this behaviour by setting this attribute. The following export hooks are recognized:

`Plain` Export all text as displayed, including escape sequences ('\33'). This is the default mode.

`EMail` Export all text, but convert the soft-style escape sequences into the pseudo-standard text sequences like:

```
*bold*      : for bold text
/italic/    : for italic text
_underline_ : for underlined text
```

```
#colored#   : for colored/highlighted text
<tsb>       : for a thick separator bar
<sb>        : for a thin separator bar
```

`NoStyle` Export all text like the `Plain` variant, but strip off all style relevant escape sequences. Also converts the escape sequences for the thick and thin separator bar into <tsb> and <sb> like the `EMail` hook.

See Section 47.23 [Texteditor.ImportHook], page 241, for details.

TYPE
String (see above for possible values)

APPLICABILITY
IS

47.18 Texteditor.ExportWrap

NAME
Texteditor.ExportWrap – use hard word wrapping

FUNCTION
This attribute allows the built-in export hooks to perform hard word wrapping while exporting text. Zero means no wrap (default value).

Please note that `Texteditor.WrapMode` doesn't have any effect on the way the export function works. That means, setting `Texteditor.ExportWrap` will always end up in hard word wrapping.

TYPE
Number

APPLICABILITY
ISG

47.19 Texteditor.FixedFont

NAME
Texteditor.FixedFont – use fixed width font

FUNCTION
Set this if you would like the editor to use a fixed width font.

TYPE
Boolean

APPLICABILITY
IG

47.20 Texteditor.GetSelection

NAME

Texteditor.GetSelection – get block selection

SYNOPSIS

```
x1, y1, x2, y2 = mui.DoMethod(id, "GetSelection")
```

FUNCTION

This method returns the start and stop position of the currently marked block. If there is no marked area, -1 is returned in all four values.

INPUTS

`id`	id of the text editor object

RESULTS

`x1`	start x position of marked block
`y1`	start y position of marked block
`x2`	stop x position of marked block
`y2`	stop y position of marked block

47.21 Texteditor.GetText

NAME

Texteditor.GetText – export portion of the current text (V1.1)

SYNOPSIS

```
t$ = mui.DoMethod(id, "GetText", x1, y1, x2, y2)
```

FUNCTION

This method exports portions of the current text and returns it. The method accepts four coordinates which permits you to retrieve a block of text without it being currently marked. This is useful to analyze a block without the user seeing a similar behaviour to marking a block.

INPUTS

`id`	id of the text editor object
`x1`	start x position of desired block
`y1`	start y position of desired block
`x2`	stop x position of desired block
`y2`	stop y position of desired block

RESULTS

`t$`	text at the specified block coordinates

47.22 Texteditor.HasChanged

NAME

Texteditor.HasChanged – learn about content change

FUNCTION

This tag will show if the contents of the gadget have changed. You can take notify on this tag, so that you can connect it with a checkmark or text object.

You should set this tag to `False` whenever you export the contents of the gadget or overwrite them with something new.

Even if you have set up notification on this tag, you should still get it before you kill the text, because this makes it possible to do some advanced testing to see if the text has actually been modified, e.g. by checking the undo buffer, comparing checksums or checking whether or not the text buffer is empty (none of this is currently done, but it may be in the future).

TYPE

Boolean

APPLICABILITY

ISGN

47.23 Texteditor.ImportHook

NAME

Texteditor.ImportHook – set text editor's import hook (V1.4)

FUNCTION

Since this gadget allows different text styles, you can supply an import hook to parse the text correctly.

The default import hook understands the following escape sequences. They may appear at any position within the line:

`\33u`	Set the soft style to underline.
`\33b`	Set the soft style to bold.
`\33i`	Set the soft style to italic.
`\33n`	Set the soft style back to normal.
`\33h`	Highlight the current line.
`\33p[x]`	Change to color `x`, where `x` is taken from the colormap. 0 means normal. The color is reset for each new line.
`\33P[RRGGBB]`	Change front color to the specified RGB color. The RGB color has to be specified in the form of six hexadecimal digits RRGGBB. This is only possible on true colour screens. The color is reset for each new line.

`\33P[AARRGGBB]`
: Change front color to the specified RGB color and apply alpha blending at the specified intensity. The RGB color has to be specified in the form of six hexadecimal digits RRGGBB which have to be prefaced with two alpha channel digits AA specifying the blending intensity. This is only possible on true colour screens. The color is reset for each new line.

The following sequences are only valid at the beginning of a line. If they are placed elsewhere, the result is undefined (they might be ignored or not):

`\33l` Left justify current and following lines.

`\33r` Right justify current and following lines.

`\33c` Center current and following lines.

`\33[s:x]` Create a separator. `x` is a bit combination of flags:

- `Bit 0 set` Top placement of separator.
- `Bit 1 set` Middle placement of separator.
- `Bit 2 set` Bottom placement of separator.
- `Bit 3 set` Draw separator on top of text (strike through).
- `Bit 4 set` Make separator extra thick.

The following values may be passed to this attribute:

`Plain` Use default import mode (see above).

`MIME` This built-in hook will convert quoted-printables (e.g. "=E5") to the ASCII representation, merge lines ending with a "=", wordwrap the text (using the value set with `TextEditor.ImportWrap`), highlight all lines that start with ">", make real *bold*, /italic/, _underline_ and #colored# text, and replace <sb> or <tsb> with a real separator bar. It will stop parsing upon reaching a `NULL` byte.

The color used for #colored# text is colormap entry 6, which defaults to `MPEN_FILL`. To override it, just supply a colormap with entry 6 set to whatever color you would like.

`MIMEQuoted`
: Like the MIME import hook, but each line gets quoted and highlighted.

`EMail` Like the MIME import hook, but it doesn't convert quoted printables.

Note, that the last three hooks also evaluate the escape sequences described for the `Plain` import hook type. While it never was documented there are some programs insisting on this side effect of an old implementation. Hence starting with version 51.13 this behaviour had to be official supported.

TYPE

String (see above for possible values)

APPLICABILITY

IS

47.24 Texteditor.ImportWrap

NAME

Texteditor.ImportWrap – perform automatic word wrapping on import

FUNCTION

This attribute allows the built-in import hooks to perform automatic word wrapping when importing text.

The built-in hooks accept a value between 4 and 1024. Default is 1023.

Please note that `Texteditor.WrapMode` doesn't have any effect on the way the import function works. That means, setting `Texteditor.ImportWrap` will always end up in hard word wrapping.

TYPE

Number

APPLICABILITY

ISG

47.25 Texteditor.Insert

NAME

Texteditor.Insert – insert text

SYNOPSIS

```
mui.DoMethod(id, "Insert", t$, pos$)
```

FUNCTION

This will insert the given text t$ at the specified position. The position of the inserted text can be one of the following:

`Cursor`	Insert at current cursor position.
`Top`	Insert at the top.
`Bottom`	Insert at the bottom.

INPUTS

`id`	id of the text editor object
`t$`	text to insert
`pos$`	insert position (see above for possible values)

47.26 Texteditor.Mark

NAME

Texteditor.Mark – mark text

SYNOPSIS

```
mui.DoMethod(id, "Mark", start_x, start_y, stop_x, stop_y)
```

FUNCTION

This method will mark (select) the given area specified by the start_x / stop_x, start_y / stop_y rectangular.

INPUTS

`id`	id of the text editor object
`start_x`	start x position
`start_y`	start y position
`stop_x`	stop x position
`stop_y`	stop y position

47.27 Texteditor.MarkAll

NAME

Texteditor.MarkAll – mark all text

SYNOPSIS

```
mui.DoMethod(id, "MarkAll")
```

FUNCTION

Marks all text in the gadget.

INPUTS

`id`	id of the text editor object

47.28 Texteditor.MarkNone

NAME

Texteditor.MarkNone – clear text selection

SYNOPSIS

```
mui.DoMethod(id, "MarkNone")
```

FUNCTION

Clears text selection.

INPUTS

`id`	id of the text editor object

47.29 Texteditor.Paste

NAME

Texteditor.Paste – paste text from clipboard

SYNOPSIS

```
mui.DoMethod(id, "Paste")
```

FUNCTION

Pastes text from clipboard into the text editor gadget.

INPUTS

`id`	id of the text editor object

47.30 Texteditor.PasteColors

NAME

Texteditor.PasteColors – ignore color information on paste

FUNCTION

Setting this tag to `False` will ignore any color information upon pasting a text clip from the clipboard.

TYPE

Boolean

APPLICABILITY

ISG

47.31 Texteditor.PasteStyles

NAME

Texteditor.PasteStyles – ignore style information on paste

FUNCTION

Setting this tag to `False` will ignore any style information upon pasting a text clip from the clipboard.

TYPE

Boolean

APPLICABILITY

ISG

47.32 Texteditor.ReadOnly

NAME

Texteditor.ReadOnly – put text editor in read-only mode

FUNCTION

Setting this tag to `True` will make the text read-only. This is very similar to what `Floattext.mui` provides, except that this gadget offers blocking.

In read-only mode:

- there will be no cursor (only a normal TAB frame),
- TAB will activate the next gadget (instead of RCOMMAND+TAB),

- the frame will be set to a ReadListFrame (may change),
- there is no ARexx support, except "Copy".

TYPE
Boolean

APPLICABILITY
ISG

47.33 Texteditor.Redo

NAME
Texteditor.Redo – redo last operation

SYNOPSIS
```
mui.DoMethod(id, "Redo")
```

FUNCTION
Redo last operation of text editor gadget.

INPUTS

`id`	id of the text editor object

47.34 Texteditor.RedoAvailable

NAME
Texteditor.RedoAvailable – learn when redo is available

FUNCTION
This tag is set to `True` when the user is able to redo his action(s) (normally after an undo). You can create a notify on this tag and let your redo button be ghosted when there is nothing to redo.

TYPE
Boolean

APPLICABILITY
GN

47.35 Texteditor.Replace

NAME
Texteditor.Replace – replace marked text with string

SYNOPSIS
```
mui.DoMethod(id, "Replace", r$)
```

FUNCTION
This method replaces the marked area with the given string.

INPUTS

`id`	id of the text editor object
`r$`	replacement string

47.36 Texteditor.Rows

NAME
Texteditor.Rows – set desired gadget height

FUNCTION
Set the desired height, in lines.

TYPE
Number

APPLICABILITY
I

47.37 Texteditor.Scrollbar

NAME
Texteditor.Scrollbar – add scrollbar to editor

FUNCTION
If you set this attribute to `True`, your texteditor object will get a scrollbar. Defaults to `False`.

TYPE
Boolean

APPLICABILITY
I

47.38 Texteditor.Search

NAME
Texteditor.Search – search for text

SYNOPSIS
```
r = mui.DoMethod(id, "Search", s$, flags$)
```

FUNCTION
Search the text for the given string. The string must not exceed 120 characters.

Flags can be a combination of:

`FromTop`	Normally the search starts at the cursor position - this flag will make it start at the beginning of the text.

`CaseSensitive`
: If you want the search to be case sensitive, then set this flag.

`Backwards`
: With this flag TextEditor will perform a backward search from cursor position.

If you specify multiple of the flags above, you have to separate them using a semicolon, e.g. "FromTop; CaseSensitive".

If the string is found, it will be automatically marked. Thus, in case you want to replace it, simply clear the marked string and insert the new one.

INPUTS

`id`	id of the text editor object
`s$`	string to search for
`flags$`	one or more of the flags listed above

RESULTS

`r`	`True` if the string was found, otherwise `False`

47.39 Texteditor.SetBold

NAME

Texteditor.SetBold – toggle bold style of text block

SYNOPSIS

```
mui.DoMethod(id, "SetBold", startx, starty, stopx, stopy, flag)
```

FUNCTION

This method toggles bold style on a text block that is defined by four coordinates (startx, starty, stopx, stopy). If the flag argument is set to `True`, bold style will be added to the specified text block, otherwise bold style will be removed from the text block.

Startx, starty, stopx, and stopy can either be absolute coordinates or one of the following special values:

`Min`	Specifies the first line or column.
`Max`	Specifies the last line or column.

INPUTS

`id`	id of the text editor object
`start_x`	start x position
`start_y`	start y position
`stop_x`	stop x position
`stop_y`	stop y position
`flag`	boolean flag that indicates toggle state

47.40 Texteditor.SetColor

NAME

Texteditor.SetColor – change color of text block (V1.2)

SYNOPSIS

```
mui.DoMethod(id, "SetColor", startx, starty, stopx, stopy, color)
```

FUNCTION

This method changes the color of a text block that is defined by four coordinates (startx, starty, stopx, stopy).

Startx, starty, stopx, and stopy can either be absolute coordinates or one of the following special values:

`Min`	Specifies the first line or column.
`Max`	Specifies the last line or column.

From the Hollywood script the color is specified as a simple numerical value containing 8 bits for each component. When you specify the color in the XML file, it has to be passed as a 6 character string prefixed by the #-character (just like in HTML).

To change the color that should be used for newly text, use the `Texteditor.Color` attribute instead.

INPUTS

`id`	id of the text editor object
`start_x`	start x position
`start_y`	start y position
`stop_x`	stop x position
`stop_y`	stop y position
`color`	desired color for text

47.41 Texteditor.SetItalic

NAME

Texteditor.SetItalic – toggle italic style of text block

SYNOPSIS

```
mui.DoMethod(id, "SetItalic", startx, starty, stopx, stopy, flag)
```

FUNCTION

This method toggles italic style on a text block that is defined by four coordinates (startx, starty, stopx, stopy). If the flag argument is set to `True`, italic style will be added to the specified text block, otherwise italic style will be removed from the text block.

Startx, starty, stopx, and stopy can either be absolute coordinates or one of the following special values:

`Min`	Specifies the first line or column.

`Max`	Specifies the last line or column.

INPUTS

`id`	id of the text editor object
`start_x`	start x position
`start_y`	start y position
`stop_x`	stop x position
`stop_y`	stop y position
`flag`	boolean flag that indicates toggle state

47.42 Texteditor.SetUnderline

NAME

Texteditor.SetUnderline – toggle underline style of text block

SYNOPSIS

```
mui.DoMethod(id, "SetUnderline", startx, starty, stopx, stopy, flag)
```

FUNCTION

This method toggles underline style on a text block that is defined by four coordinates (startx, starty, stopx, stopy). If the flag argument is set to `True`, underline style will be added to the specified text block, otherwise underline style will be removed from the text block.

Startx, starty, stopx, and stopy can either be absolute coordinates or one of the following special values:

`Min`	Specifies the first line or column.
`Max`	Specifies the last line or column.

INPUTS

`id`	id of the text editor object
`start_x`	start x position
`start_y`	start y position
`stop_x`	stop x position
`stop_y`	stop y position
`flag`	boolean flag that indicates toggle state

47.43 Texteditor.StyleBold

NAME

Texteditor.StyleBold – set/get bold style

FUNCTION

This tag shows whether the cursor or block is over bolded text or not. You can set up a notification on this tag to learn about style changes. You can set this tag to `True` or `False` if you want the style changed.

TYPE

Boolean

APPLICABILITY

SGN

47.44 Texteditor.StyleItalic

NAME

Texteditor.StyleItalic – set/get italic style

FUNCTION

This tag shows whether the cursor or block is over italics text or not. You can set up a notification on this tag to learn about style changes. You can set this tag to `True` or `False` if you want the style changed.

TYPE

Boolean

APPLICABILITY

SGN

47.45 Texteditor.StyleUnderline

NAME

Texteditor.StyleUnderline – set/get underline style

FUNCTION

This tag shows whether the cursor or block is over underlined text or not. You can set up a notification on this tag to learn about style changes. You can set this tag to `True` or `False` if you want the style changed.

TYPE

Boolean

APPLICABILITY

SGN

47.46 Texteditor.TabSize

NAME

Texteditor.TabSize – set/get tab size

FUNCTION

This tag overrides the global TAB size as configured by MUI prefs if a specific number of spaces per TAB needs to be enforced. Valid values range from 2 to 12 spaces per TAB character. You can pass the special value `Default` to reset the value to the user configured value.

Use this attribute with care as it contradicts the MUI philosophy to let the user choose the settings.

TYPE

Number or string (see above)

APPLICABILITY

ISG

47.47 Texteditor.Undo

NAME

Texteditor.Undo – undo last operation

SYNOPSIS

```
mui.DoMethod(id, "Undo")
```

FUNCTION

Undo last operation of text editor gadget.

INPUTS

`id`	id of the text editor object

47.48 Texteditor.UndoAvailable

NAME

Texteditor.UndoAvailable – learn when undo is available

FUNCTION

This tag is set to `True` when the user is able to undo his action(s). You can create a notify on this tag and let your undo button be ghosted when there is nothing to undo.

TYPE

Boolean

APPLICABILITY

GN

47.49 Texteditor.UndoLevels

NAME

Texteditor.UndoLevels – set/get available undo levels

FUNCTION

Using this tag an application is able to override the global undo level setting. A value of zero will disable undo/redo completely. Upon get the current number of undo levels will be returned.

TYPE

Number

APPLICABILITY

ISG

47.50 Texteditor.WrapBorder

NAME

Texteditor.WrapBorder – set auto word wrap border

FUNCTION

This attribute allows to define the number of characters in texteditor at which text should be wrapped to the next line (i.e. 'autowrap').

However, the way wrapping is done depends on the actual wrap mode that was specified by `Texteditor.WrapMode`. In addition, `Texteditor.WrapWords` will allow to set if text editor will wrap at word boundaries or not.

Per default the WrapBorder is set to 0 which in fact disables the wrapping and depending on the set wrap mode, wrapping will either be completely ignored or wrapping will be done at the window limits, which is the default.

TYPE

Number

APPLICABILITY

ISG

47.51 Texteditor.WrapMode

NAME

Texteditor.WrapMode – set/get wrap mode

FUNCTION

With this attribute, the way text wrapping is performed can be directly controlled if `Texteditor.WrapBorder` was set `>` 0. There are two major ways of how wrapping can be performed. 'soft wrapping' and 'hard wrapping'. Whereas soft wrapping refers to the process where lines will just be graphically wrapped to the next line, but are still be considered a single line. Hard wrapping in turn immediately inserts a newline character as soon as the cursor passed the specified amount of characters in one line and therefore separate them in a 'hard' direct way.

The following modes are supported:

`NoWrap` Wrapping is completely disabled, setting WrapBorder `>` 0 will have no effect. That means, the window will have to be resizes by the user to e.g. see or edit all text.

`SoftWrap` Enables soft wrapping of lines if WrapBorder `>` 0. That means, lines are being wrapped at the specified border in a way that they are still logically one line. This allows to e.g. dynamically reconcatenate lines. This should be considered the new preferred setting for new/revised applications. If WrapBorder == 0, soft wrapping will be performed at the current window limits.

`HardWrap` Enables hard wrapping of lines if WrapBorder `>` 0. That means, lines are being wrapped at the specified border by directly inserting newline characters as soon as the cursor passes the specified border. While this mode is the default, due to historical reasons, it doesn't allow to directly reconcatenate lines if the user e.g. removes characters. If WrapBorder == 0, soft wrapping will be performed at the current window limits. This is the default

Due to historical reasons the default is to hard wrap a line if the WrapBorder attribute is set `>` 0 and WrapMode is kept untouched. However, for new applications, soft wrapping should be considered the preferred setting as this mode is more intuitive and allows users of texteditor to directly concatenate lines if they e.g. remove characters while they are writing.

TYPE

String (see above for possible values)

APPLICABILITY

ISG

47.52 Texteditor.WrapWords

NAME

Texteditor.WrapWords – wrap only between words

FUNCTION

When set to `True` (default), wrapping will occur at word boundaries (e.g. space or tab). If `False` text editor will wrap at any character.

Change of this attribute will cause update of editor gadget (but cursor coordinates will be reset to zeroes).

TYPE

Boolean

APPLICABILITY

ISG

48 Toolbar class

48.1 Overview

Toolbar class allows you to create toolbar gadgets for your windows. Toolbars can be displayed either as images, images and text, or text only. This class requires the MUI custom class `TheBar.mcc`.

`TheBar.mcc` is Copyright (C) 2003-2005 Alfonso Ranieri and Copyright (C) 2005-2009 TheBar Open Source Team

It is released and distributed under the terms of the GNU Lesser General Public License (LGPL) and available free of charge.

Please visit `http://www.sf.net/projects/thebar/` for the very latest version and information regarding `TheBar.mcc`

When declaring a toolbar gadget in XML code, you always need to add at least one toolbar button to it. This is done using Toolbarbutton class. Here is an example XML declaration of a toolbar with three buttons:

```
<toolbar horiz="true">
   <button image="1">Cut</button>
   <button image="2">Copy</button>
   <button image="3">Paste</button>
</toolbar>
```

In the XML declaration above, toolbar button 1 will use Hollywood brush 1 as its image, button 2 will use brush 2, and so on. Toolbar buttons can use many more options like special images for selected and disabled states, help bubbles, and more. See Section 49.1 [Toolbarbutton class], page 271, for details.

48.2 Toolbar.Active

NAME

Toolbar.Active – set/get active button in mutual exclude group

FUNCTION

This attributes can be used to query or set the ID of the currently active button in the toolbar with mutual excluding buttons involved.

Note that you may have many mutual exclude groups: this attribute will always contain the ID of the last selected button.

TYPE

Number

APPLICABILITY

ISGN

48.3 Toolbar.BarPos

NAME

Toolbar.BarPos – set button alignment inside toolbar

FUNCTION

This attributes defines the alignment of the buttons in a bar. This can be one of:

`Left`	Left alignment. This is also the default.
`Center`	Centered alignment.
`Right`	Right alignment.

TYPE

String (see above for possible values)

APPLICABILITY

ISG

48.4 Toolbar.BarSpacer

NAME

Toolbar.BarSpacer – turn spaces into bar spacers

FUNCTION

If this attribute is `True`, any space spacer becomes a bar spacer. Defaults to `False`

TYPE

Boolean

APPLICABILITY

ISG

48.5 Toolbar.BarSpacerSpacing

NAME

Toolbar.BarSpacerSpacing – set bar spacer spacing

FUNCTION

Defines the pixels between a bar spacer and the buttons at its left/right. Accepted range is 0<=x<=16

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Number

APPLICABILITY

I

48.6 Toolbar.Borderless

NAME

Toolbar.Borderless – make toolbar borderless

FUNCTION

If this attribute is `True`, you get borderless buttons. Defaults to `False`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

ISG

48.7 Toolbar.BottomBarFrameSpacing

NAME

Toolbar.BottomBarFrameSpacing – set bottom spacing

FUNCTION

Define the pixels between the bar and the bottom frame. Accepted range is 0<x<=16

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Number

APPLICABILITY

I

48.8 Toolbar.BottomInnerSpacing

NAME

Toolbar.BottomInnerSpacing – set bottom inner spacing

FUNCTION

Define the pixels between a button contents and its bottom frame. Accepted range is 0<x<=16

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

I

48.9 Toolbar.Columns

NAME

Toolbar.Columns – set/get toolbar columns

FUNCTION

This attributes defines the number of the columns of the bar. Setting this attribute to a value greater than 0, makes the bar act in a so called `"columns mode"`. Defaults to 0.

TYPE

Number

APPLICABILITY

ISG

48.10 Toolbar.DisMode

NAME

Toolbar.DisMode – set disabled draw mode

FUNCTION

Allows to define how disabled buttons should be rendered in case no own image set is available for it.

The following values are possible:

`Shape`	Draws only a B/W shape to show the disabled status.
`Grid`	Draw a light grid in front of the image.
`FullGrid`	Draw a full grid.
`Sunny`	Converts the image into a greyscale image and displays it.
`Blend`	The disabled image is somehow blended. It requires a true color image with an active alpha channel and a system capable to render it.
`BlendGray`	As the above, but rendered as gray.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

String (see above for possible values)

APPLICABILITY

ISG

48.11 Toolbar.DontMove

NAME

Toolbar.DontMove – create static toolbar

FUNCTION

If this attribute is `True`, the content of the buttons is not moved when they are active. Defaults to `False`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

I

48.12 Toolbar.EnableKeys

NAME

Toolbar.EnableKeys – enable key shortcuts

FUNCTION

If this attribute is set to `True`, the key shortcuts are enabled. Key shortcuts are defined by prepending the key with a `"_"` in the button text. Defaults to `True`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

ISG

48.13 Toolbar.Frame

NAME

Toolbar.Frame – draw frame around toolbar

FUNCTION

If this attribute is `True`, a frame is drawn around the bar. Defaults to `False`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

ISG

48.14 Toolbar.Free

NAME

Toolbar.Free – allow resizing of toolbar

FUNCTION

If this attribute is `True`, the bar is allowed to grow freely in both, vertical and horizontal direction.

The default is:

- If `Toolbar.Horiz` is `True` the bar can freely grow in horizontal direction, but is limited vertically.
- If `Toolbar.Horiz` is `False` the can freely grow in vertical direction, but is limited horizontal.

TYPE

Boolean

APPLICABILITY

ISG

48.15 Toolbar.FreeHoriz

NAME

Toolbar.FreeHoriz – allow horizontal resizing of toolbar

FUNCTION

If this attribute is `True`, the bar is allowed to grow freely in horizontal direction.

The default is:

- If `Toolbar.Horiz` is `True` the bar can freely grow in horizontal direction, but is limited vertically.
- If `Toolbar.Horiz` is `False` the can freely grow in vertical direction, but is limited horizontal.

TYPE

Boolean

APPLICABILITY

ISG

48.16 Toolbar.FreeVert

NAME

Toolbar.FreeVert – allow vertical resizing of toolbar

FUNCTION

If this attribute is `True`, the bar is allowed to grow freely in vertical direction.

The default is:

- If `Toolbar.Horiz` is `True` the bar can freely grow in horizontal direction, but is limited vertically.

- If `Toolbar.Horiz` is `False` the can freely grow in vertical direction, but is limited horizontal.

TYPE
Boolean

APPLICABILITY
ISG

48.17 Toolbar.Horiz

NAME
Toolbar.Horiz – set toolbar orientation

FUNCTION
Boolean value to indicate whether the toolbar buttons shall be layouted horizontally or vertically. Defaults to `False`.

TYPE
Boolean

APPLICABILITY
I

48.18 Toolbar.HorizInnerSpacing

NAME
Toolbar.HorizInnerSpacing – set horiz inner spacing

FUNCTION
Define the pixels between a button contents and its left and right frames. Accepted range is 0 < x <= 16

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE
Number

APPLICABILITY
I

48.19 Toolbar.HorizSpacing

NAME
Toolbar.HorizSpacing – set horiz spacing

FUNCTION
Define the pixels between two bar columns. Accepted range is 0 < x <= 16

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Number

APPLICABILITY

I

48.20 Toolbar.HorizTextGfxSpacing

NAME

Toolbar.HorizTextGfxSpacing – set horiz text gfx spacing

FUNCTION

Define the pixels between the image and the Left/Right label in Text/Gfx buttons. Accepted range is 0 < x <= 16.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Number

APPLICABILITY

I

48.21 Toolbar.IgnoreAppearance

NAME

Toolbar.IgnoreAppearance – ignore user preferences

FUNCTION

The user is able to change the general appearance of the object in the MUI preferences.

If this attribute is `True`, MCP appearance preferences are ignored. If this attribute is `False`, MCP appearance preferences are used.

Defaults to `False`.

TYPE

Boolean

APPLICABILITY

ISG

48.22 Toolbar.LabelPos

NAME

Toolbar.LabelPos – set position of the text label

FUNCTION

Controls the position of the text in a Text/Gfx toolbar. The following positions are supported:

`Bottom`	Text appears below the button image. This is the default.

`Top`	Text appears above the button image.
`Right`	Text appears to the right of the button image.
`Left`	Text appears to the left of the button image.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE
String (see above for possible values)

APPLICABILITY
ISG

48.23 Toolbar.LeftBarFrameSpacing

NAME
Toolbar.LeftBarFrameSpacing – set left bar frame spacing

FUNCTION
Define the pixels between the bar and its left frame. Accepted range is 0 `<` x `<=` 16.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE
Number

APPLICABILITY
I

48.24 Toolbar.MouseOver

NAME
Toolbar.MouseOver – get informed about mouse over event

FUNCTION
If `Toolbar.Raised` or `Toolbar.Sunny` is `True`, this attribute contains the id of the last button where the mouse was over.

TYPE
Number

APPLICABILITY
N

48.25 Toolbar.NtRaiseActive

NAME

Toolbar.NtRaiseActive – remove frame for raised buttons

FUNCTION

If `False`, a frame is drawn around borderless and sunny or raised buttons. If `True`, no frame is used.

Useful to activate a button when the mouse is over and so change page in a page-mode group.

Defaults to `False`.

TYPE

Boolean

APPLICABILITY

ISG

48.26 Toolbar.Raised

NAME

Toolbar.Raised – use raised mode

FUNCTION

If `True`, a frame is drawn around a borderless button when the mouse is over it. It should be set to `False` when `Toolbar.Borderless` is `False`. Defaults to `False`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

ISG

48.27 Toolbar.RightBarFrameSpacing

NAME

Toolbar.RightBarFrameSpacing – set right bar frame spacing

FUNCTION

Define the pixels between the bar and its right frame. Accepted range is 0 < x <= 16.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Number

APPLICABILITY

I

48.28 Toolbar.Rows

NAME

Toolbar.Rows – set/get toolbar rows

FUNCTION

This attribute defines the number of rows of the toolbar should be layouted with. Setting this attribute to a value greater than 0 make the toolbar running in a so-called rows mode. It will the act as vertically unlimited. Defaults to 0.

TYPE

Number

APPLICABILITY

ISG

48.29 Toolbar.Scale

NAME

Toolbar.Scale – set scale ratio

FUNCTION

Define the scale ratio for scaled buttons. Accepted range is 50 <= x <= 200

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Number

APPLICABILITY

ISG

48.30 Toolbar.Scaled

NAME

Toolbar.Scaled – enable button scaling

FUNCTION

If `True`, all buttons will be scaled down according to defined scale ratio. Defaults to `False`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

ISG

48.31 Toolbar.SpacersSize

NAME

Toolbar.SpacersSize – define spacer size

FUNCTION

Defines the used size of spacers relative to the size of a button. The following values are possible:

`Quarter`	Quarter of a button. This is the default.
`Half`	Half of a button.
`One`	Same as a button.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

String (see above for possible values)

APPLICABILITY

ISG

48.32 Toolbar.SpecialSelect

NAME

Toolbar.SpecialSelect – shift button contents on select

FUNCTION

If `True`, the buttons contents is moved to the right/bottom when the mouse is over it. Defaults to `False`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

I

48.33 Toolbar.Sunny

NAME

Toolbar.Sunny – use sunny mode

FUNCTION

If `True`, buttons are rendered in black/white, but when the mouse is over them they are rendered colored. Defaults to `False`.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

ISG

48.34 Toolbar.TextOverUseShine

NAME

Toolbar.TextOverUseShine – use shine pen for highlighted state

FUNCTION

If `True`, buttons labels are rendered with the shine rather that the text pen, when the mouse is over them.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Boolean

APPLICABILITY

I

48.35 Toolbar.TopBarFrameSpacing

NAME

Toolbar.TopBarFrameSpacing – set top spacing

FUNCTION

Define the pixels between the bar and its top frame. Accepted range is 0<x<=16

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

Number

APPLICABILITY

I

48.36 Toolbar.TopInnerSpacing

NAME

Toolbar.TopInnerSpacing – set top inner spacing

FUNCTION

Define the pixels between a button contents and its top frame. Accepted range is 0 < x <= 16.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE
Boolean

APPLICABILITY
I

48.37 Toolbar.VertSpacing

NAME
Toolbar.VertSpacing – set vertical spacing

FUNCTION
Defines the pixels between two bar rows. Accepted range is 0 < x <= 16.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE
Number

APPLICABILITY
I

48.38 Toolbar.VertTextGfxSpacing

NAME
Toolbar.VertTextGfxSpacing – set vertical text gfx spacing

FUNCTION
Defines the pixels between the image and the top/bottom placed label in text/gfx buttons. Accepted range is 0 < x <= 16.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE
Number

APPLICABILITY
I

48.39 Toolbar.ViewMode

NAME
Toolbar.ViewMode – set view mode of toolbar

FUNCTION
Sets the style of the toolbar. The following styles are possible:

`TextGfx`	Toolbar appears as text and images. This is also the default setting.

`Gfx`	Toolbar appears as images only.
`Text`	Toolbar appears as text only.

This overwrites the user preferences and so must be used wisely. If in doubt, don't use it.

TYPE

String (see above for possible values)

APPLICABILITY

ISG

49 Toolbarbutton class

49.1 Overview

Toolbar button class is a subclass of toolbar class. It cannot be used on its own but must always be encapsulated inside a toolbar class definition. Toolbar button class creates a single button for its super toolbar class. Please note that the XML tag for this class is just `<button>`, not `<toolbarbutton>`. See Section 48.1 [Toolbar class], page 255, for details.

49.2 Toolbarbutton.Disabled

NAME
Toolbarbutton.Disabled – enable/disable button

FUNCTION
Enable/disable button.

TYPE
Boolean

APPLICABILITY
ISG

49.3 Toolbarbutton.DisImage

NAME
Toolbarbutton.DisImage – set disabled button image

FUNCTION
Sets the image for this toolbar button that shall be displayed when the button is in disabled state. This must be set to the identifier of a Hollywood brush.

TYPE
Number

APPLICABILITY
I

49.4 Toolbarbutton.Exclude

NAME
Toolbarbutton.Exclude – set exclusion mask

FUNCTION
Allows to specify a mutual exlusive mask to link up to 24 button together. The mask must be specified as a bit operation of the type 1<<x where x = 0,...,23 is the ID of the button it the mutual exclude mask.

TYPE

Number

APPLICABILITY

I

49.5 Toolbarbutton.Hide

NAME

Toolbarbutton.Hide – show/hide button

FUNCTION

Use this attribute to show/hide toolbar buttons.

TYPE

Boolean

APPLICABILITY

ISG

49.6 Toolbarbutton.Image

NAME

Toolbarbutton.Image – set button image

FUNCTION

Sets the image for this toolbar button. This must be set to the identifier of a Hollywood brush or one of the following special values:

`BarSpacer`	Create a bar spacer.
`ButtonSpacer`	Create a spacer between two buttons.

TYPE

Number or string (see above for possible values)

APPLICABILITY

I

49.7 Toolbarbutton.Immediate

NAME

Toolbarbutton.Immediate – create immediate button

FUNCTION

If you set this to `True`, the button will be an immediate one.

TYPE
Boolean

APPLICABILITY
I

49.8 Toolbarbutton.NoClick

NAME
Toolbarbutton.NoClick – create a dead button

FUNCTION
If you set this to `True`, the button won't react to user inputs.

TYPE
Boolean

APPLICABILITY
I

49.9 Toolbarbutton.Pressed

NAME
Toolbarbutton.Pressed – learn if a button is pressed

FUNCTION
This attribute is triggered if the user presses the button.

TYPE
Boolean

APPLICABILITY
N

49.10 Toolbarbutton.Selected

NAME
Toolbarbutton.Selected – toggle selection state

FUNCTION
Use this to toggle the selection state of the button or get notified of a user toggle event.

TYPE
Boolean

APPLICABILITY
ISGN

49.11 Toolbarbutton.SelImage

NAME

Toolbarbutton.SelImage – set selected button image

FUNCTION

Sets the image for this toolbar button that shall be displayed when the button is in selected state. This must be set to the identifier of a Hollywood brush.

TYPE

Number

APPLICABILITY

I

49.12 Toolbarbutton.ShortHelp

NAME

Toolbarbutton.ShortHelp – set short help for toolbar button

FUNCTION

Sets the button's short-help (for bubble help).

The string you specify here can use text formatting codes. See Section 3.9 [Text formatting codes], page 14, for details.

TYPE

String

APPLICABILITY

I

49.13 Toolbarbutton.Sleep

NAME

Toolbarbutton.Sleep – create sleeping button

FUNCTION

If this is set to `True`, the button is not created at all.

TYPE

Boolean

APPLICABILITY

ISG

49.14 Toolbarbutton.Toggle

NAME

Toolbarbutton.Toggle – create toggle button

FUNCTION

If you set this to `True`, the button will be a toggle button.

TYPE

Boolean

APPLICABILITY

I

50 Virtgroup class

50.1 Overview

Virtgroup class generates special kinds of group objects whose children can be a lot larger than the actual group. The group acts as a (small) window through which a rectangle area of its contents is visible.

During layout, MUI tries to place the children of a virtual group in the visible part. If this is impossible, space is extended as long as all children fit.

Virtual groups themselves don't offer any scrollbars to allow user interaction. These things are handled by scrollgroup class. Usually, you don't want to use a virtual group without a scrollgroup. See Section 43.1 [Scrollgroup class], page 217, for details.

50.2 Virtgroup.Height

NAME

Virtgroup.Height – get height of virtual group

FUNCTION

Read the virtual height of a virtual group. This attribute is quite senseless, better use a scrollgroup object to control the virtual group.

TYPE

Number

APPLICABILITY

G

50.3 Virtgroup.Horiz

NAME

Virtgroup.Horiz – set layout mode

FUNCTION

Boolean value to indicate whether the objects in this group shall be layouted horizontally or vertically. Defaults to `False`.

This is the easy way of telling your group how it has to look like. If you want two-dimensional groups, you have to use `Group.Columns`.

TYPE

Boolean

APPLICABILITY

I

50.4 Virtgroup.Input

NAME

Virtgroup.Input – set input mode of virtual group

FUNCTION

Specify if a virtual group should be moveable by clicking into it and dragging the mouse. Defaults to `True`.

TYPE

Boolean

APPLICABILITY

I

50.5 Virtgroup.Left

NAME

Virtgroup.Left – set/get left edge of virtual group

FUNCTION

Get/set the virtual left edge of a virtual group. The left edge will automatically be clipped to be between 0 and (VirtualWidth-DisplayWidth).

This attribute is quite senseless, better use a scrollgroup object to control the virtual group.

TYPE

Number

APPLICABILITY

ISG

50.6 Virtgroup.Top

NAME

Virtgroup.Top – set/get top edge of virtual group

FUNCTION

Get/set the virtual top edge of a virtual group. The top edge will automatically be clipped to be between 0 and (VirtualTop-DisplayTop).

This attribute is quite senseless, better use a scrollgroup object to control the virtual group.

TYPE

Number

APPLICABILITY

ISG

50.7 Virtgroup.Width

NAME

Virtgroup.Width – set/get width of virtual group

FUNCTION

Read the virtual width of a virtual group. This attribute is quite senseless, better use a scrollgroup object to control the virtual group.

TYPE

Number

APPLICABILITY

G

51 Volumelist class

51.1 Overview

Volumelist generates a list of all available volumes. Since you shouldn't use your own file requester in every application, this class is probably not of much use.

Volumelist class creates a listview with the following five columns: Icon, name, fill state in percent, free space and used space. If you do not want to have all of these attributes displayed, you can use Listviewcolumn class to make adjustments.

Volumelist class is a subclass of listview class. Thus, you can use most attributes and methods of listview class on it too. For example, if you want to read the entries of your volumelist, just send the volumelist object a `Listview.GetEntry` method, or if you want to listen to changes in the active list entry, set up a notification on `Listview.Active`.

When creating a volumelist object in XML code, you always have to add at least one column to it. This is done by using the Listviewcolumn class. Here is an example of a minimal volumelist declaration with just a single column (the icon column, that is):

```
<volumelist>
   <column/>  <!-- Icon -->
</volumelist>
```

Here is a declaration that includes all five columns:

```
<volumelist>
   <column/>  <!-- Icon -->
   <column/>  <!-- Name -->
   <column/>  <!-- Percent -->
   <column/>  <!-- Free -->
   <column/>  <!-- Used -->
</volumelist>
```

If you only want to have the name column and the percent column displayed, you can use the `Listviewcolumn.Col` attribute to achieve this:

```
<volumelist>
   <column col="1"/>  <!-- Name -->
   <column col="2"/>  <!-- Percent -->
</volumelist>
```

See Section 25.1 [Listview class], page 143, for details.

See Section 26.1 [Listviewcolumn class], page 159, for details.

51.2 Volumelist.Title

NAME

Volumelist.Title – show/hide list title

FUNCTION

Specify whether you want to have title bar for the listview. The title is displayed at the very first line and doesn't scroll away when the list top position moves.

TYPE
Boolean

APPLICABILITY
ISG

52 VSpace class

52.1 Overview

VSpace class is a subclass of rectangle class and simply creates some empty vertical space between two objects.

52.2 VSpace.Height

NAME
VSpace.Height – set vertical space

FUNCTION
Sets the desired vertical space for this object in pixels.

TYPE
Number

APPLICABILITY
I

53 Window class

53.1 Overview

Objects of window class are used to generate windows and supply a place where MUI gadgets feel well. It handles the complicated task of window resizing fully automatic, you don't need to worry about that.

Windows are children of an application, you cannot use a window object without having a parent application object. On the other side, the gadgets in a window are children of the window, you cannot use MUI gadgets without having a parent MUI window.

Creating a window object does not mean to open it instantly. This is done later by setting the window's `Window.Open` attribute. If your application has several windows, the usual way is to create them all at once at startup time and open/close it later just by setting `Window.Open`.

There is no difference in talking to gadgets whether their parent window is open or not. If you e.g. set the contents of a string gadget in an open window, the gadget will refresh immediately. If the window is closed, the gadget just remembers its new setting and displays it later.

53.2 Window.Activate

NAME

Window.Activate – change activation state of window

FUNCTION

Setting this to `True` will activate the window. Setting this to `False` has no effect. The attribute will change whenever the user activates/deactivates the window.

Specifying `False` at object creation time will make the window open in an inactive state.

TYPE

Boolean

APPLICABILITY

ISGN

53.3 Window.ActiveObject

NAME

Window.ActiveObject – set/get active window object

FUNCTION

Set the active object in a window as if the user would have activated it with the tab key. The object has to be in the cycle chain for this command to work.

Starting with MUI Royale 1.1 the applicability of this attribute is SGN. Before version 1.1 it was merely SG.

TYPE

MUI object

APPLICABILITY

SGN

53.4 Window.AppWindow

NAME

Window.AppWindow – make window an app window

FUNCTION

Setting this attribute to `True` will make this window an AppWindow, the user will be able to drop icons on it. You can hear about these events by listening to the `Notify.AppMessage` attribute.

TYPE

Boolean

APPLICABILITY

I

53.5 Window.Borderless

NAME

Window.Borderless – show/hide window border

FUNCTION

Make the window borderless.

TYPE

Boolean

APPLICABILITY

I

53.6 Window.CloseGadget

NAME

Window.CloseGadget – configure window's close gadget

FUNCTION

Set this to `False` and your window will not have a close gadget.

TYPE

Boolean

APPLICABILITY

I

53.7 Window.CloseRequest

NAME

Window.CloseRequest – handle close request of window

FUNCTION

When the user hits a windows close gadget, the window isn't closed immediately. Instead MUI only sets this attribute to `True` to allow your application to react.

Usually, you will setup a notification that automatically closes the window when a close request appears, but you could e.g. pop up a confirmation requester or do some other things first.

TYPE

Boolean

APPLICABILITY

N

53.8 Window.DefaultObject

NAME

Window.DefaultObject – set/get window's default object

FUNCTION

The default object in a window receives keyboard input as long as no other object is active. Good candidates for default objects are e.g. lonely listviews. Making such a listview the default object will allow the user to control it immediately without the need of several tab strokes for activation.

TYPE

MUI object

APPLICABILITY

ISG

53.9 Window.DepthGadget

NAME

Window.DepthGadget – configure window's depth gadget

FUNCTION

Enable or disable the depth gadget. Defaults to `True`. There is no good reason to use this tag.

TYPE

Boolean

APPLICABILITY

I

53.10 Window.DragBar

NAME

Window.DragBar – configure window's drag bar

FUNCTION

Tell MUI to give your window a dragbar.

Defaults to `True`.

There is no good reason to disable the dragbar!

TYPE

Boolean

APPLICABILITY

I

53.11 Window.Height

NAME

Window.Height – set/get window height

FUNCTION

Specify the height of a window. Usually, you won't give a pixel value here but instead use one of the following magic macros:

`Default`	Calculated from objects default sizes.
`MinMax:<0..100>`	Somewhere between the minimum height (0) and the maximum height (100) of your window.
`Visible:<1..100>`	Percentage of the screens visible height.
`Screen:<1..100>`	Percentage of the screens total height.
`Scaled`	Height will be adjusted so that width : height == minimum width : minimum height. Note that a windows width and height may not both be scaled.

Default for this tag is `Default`.

As long as your window has a MUI window id (`Window.MuiID`), choosing a size is not that important. MUI will always remember a windows last position and size and these values will simply override your settings. Positioning and sizing should be completely under user control, a programmer doesn't need to worry about it.

TYPE

Number or predefined macro

APPLICABILITY

IG

53.12 Window.LeftEdge

NAME

Window.LeftEdge – set/get left edge of window

FUNCTION

Specify the left edge of a window. Usually, you shouldn't define a pixel value here but instead use one of the following macros:

`Centered`	Window appears centered on the visible area of screen.
`Moused`	Window appears centered under the mouse pointer.

Default for this tag is `Centered`.

As long as your window has a window id (`Window.MuiID`), choosing a position is not that important. MUI will always remember a windows last position and size and these values will simply override your settings. Positioning and sizing should be completely under user control, a programmer doesn't need to worry about it.

TYPE

Number or predefined macro

APPLICABILITY

IG

53.13 Window.Menustrip

NAME

Window.Menustrip – set window's menustrip

FUNCTION

Specify a menu strip object for this window. The object is treated as a child of the window and will be disposed when the window is disposed.

Menustrip objects defined for a window will override an applications Menustrip object.

If you have a global menu for all your applications windows but you want some windows to have no menu, use the `Window.NoMenus` tag.

TYPE

MUI object

APPLICABILITY

I

53.14 Window.MouseObject

NAME

Window.MouseObject – find object that mouse is currently over

FUNCTION

When `Window.NeedsMouseObject` is enabled for this window, you can setup notificationns on `Window.MouseObject` to find out on which object the mouse pointer is located.

TYPE

MUI object

APPLICABILITY

N

53.15 Window.MuiID

NAME

Window.MuiID – set/get window identifier

FUNCTION

For most of your windows, you should define a four character string as an id value. Only a window with an id is able to remember its size and position.

Additionally, when you use an ascii id (e.g. 'MAIN'), your window can be controlled from ARexx.

Of course all windows of your application must have unique ids.

Do not confuse `Window.MuiID` with `Notify.ID`. The ID you specify in `Notify.ID` is the one you use to talk to MUI objects using commands like `mui.Set()` or `mui.Get()`. It does not have anything to do with the ID you set in `Window.MuiID`!

TYPE

Four character string

APPLICABILITY

ISG

53.16 Window.NeedsMouseObject

NAME

Window.NeedsMouseObject – enable mouse tracking

FUNCTION

If you want to react on changes of the `Window.MouseObject` attribute, you have to set this to `True` when creating your window.

TYPE

Boolean

APPLICABILITY

I

53.17 Window.NoMenus

NAME

Window.NoMenus – disable window menu

FUNCTION
Temporarily disable the menu strip of a window.

TYPE
Boolean

APPLICABILITY
IS

53.18 Window.Open

NAME
Window.Open – open/close a window

FUNCTION
This little attribute can be used to open and close a window. When opening a window, MUI does lots of stuff to calculate sizes and positions of all gadgets. Minimum and maximum window sizes will be adjusted automatically.

When the minimum size of a window is too big to fit on the screen, MUI tries to reduce font sizes and does a new calculation. You should always design your windows to fit on a 640*200 screen with all fonts set to topaz/8.

When a window is closed (and you specified a `Window.MuiID`, MUI remembers its position and size and uses these values during the next opening.

After setting `Window.Open` to `True`, you should test if MUI was able to open the window by getting the attribute again. If you don't and if this was the only window of your application, the user won't be able to do any input and your application will seem to hang.

TYPE
Boolean

APPLICABILITY
SG

53.19 Window.PubScreen

NAME
Window.PubScreen – set window's screen (V1.1)

FUNCTION
This attribute allows you to specify the name of a public screen that the window should open on. Please use this attribute only if really necessary because normally the user of your application should be the one who decides on which screen he wants to run your application using the MUI preferences.

TYPE
String

APPLICABILITY
ISG

53.20 Window.ScreenTitle

NAME
Window.ScreenTitle – set screen title of window

FUNCTION
This text will appear in the screen's title bar when the window is active.

TYPE
String

APPLICABILITY
ISG

53.21 Window.ScreenToBack

NAME
Window.ScreenToBack – put window's screen to back

SYNOPSIS
```
mui.DoMethod(id, "ScreenToBack")
```

FUNCTION
Put the window's screen to back. This command is only valid when the window is opened.

INPUTS

`id`	id of the window object

53.22 Window.ScreenToFront

NAME
Window.ScreenToFront – put window's screen to front

SYNOPSIS
```
mui.DoMethod(id, "ScreenToFront")
```

FUNCTION
Put the window's screen to font. This command is only valid when the window is opened.

INPUTS

`id`	id of the window object

53.23 Window.Sleep

NAME

Window.Sleep – toggle sleep mode of window

FUNCTION

This attribute can be used to put a window to sleep. The window gets disabled and a busy pointer appears.

The attribute contains a nesting count, if you tell your window to sleep twice, you will have to tell it to wake up twice too.

A sleeping window cannot be resized.

TYPE

Boolean

APPLICABILITY

SG

53.24 Window.SizeGadget

NAME

Window.SizeGadget – configure window's size gadget

FUNCTION

Tell MUI if you want a sizing gadget for this window. Usually you won't need this attribute since MUI will automatically disable the sizing gadget when your window is not sizeable because of your gadget layout.

TYPE

Boolean

APPLICABILITY

I

53.25 Window.Snapshot

NAME

Window.Snapshot – snapshot window position

SYNOPSIS

```
mui.DoMethod(id, "Snapshot", flags)
```

FUNCTION

`Window.Snapshot` is the programmer's interface to MUI's window position remembering facility. Snapshotting a window is only possible if `Window.MuiID` was set for this window.

INPUTS

`id`	id of the window object
`flags`	use 0 to unsnapshot the window, 1 to snapshot the window.

53.26 Window.ToBack

NAME

Window.ToBack – put window to back

SYNOPSIS

```
mui.DoMethod(id, "ToBack")
```

FUNCTION

Put the window to back. When the window is not currently open, this command does simply nothing.

INPUTS

`id`	id of the window object

53.27 Window.ToFront

NAME

Window.ToFront – put window to front

SYNOPSIS

```
mui.DoMethod(id, "ToFront")
```

FUNCTION

Put the window to front. When the window is not currently open, this command does simply nothing.

INPUTS

`id`	id of the window object

53.28 Window.TopEdge

NAME

Window.TopEdge – set/get top edge of window

FUNCTION

Specify the top edge of a window. Usually, you shouldn't define a pixel value here but instead use one of the following macros:

`Centered`	Window appears centered on the visible area of screen.
`Moused`	Window appears centered under the mouse pointer.
`Delta:<p>`	Window appears `<p>` pixels below the screens title bar.

Default for this tag is `Centered`.

As long as your window has a window id (`Window.MuiID`), choosing a position is not that important. MUI will always remember a windows last position and size and these values will simply override your settings. Positioning and sizing should be completely under user control, a programmer doesn't need to worry about it.

TYPE

Number or predefined macro

APPLICABILITY

IG

53.29 Window.UseBottomBorderScroller

NAME

Window.UseBottomBorderScroller – enable bottom border scrollbar

FUNCTION

If set to `True`, the window will feature a scrollbar in its bottom border. You must set this for the window object if any children are going to use this window border scroller, e.g. prop gadgets with the `Prop.UseWinBorder` attribute.

Obviously, scroll gadgets in window borders won't look good with borderless or non-resizable windows.

TYPE

Boolean

APPLICABILITY

I

53.30 Window.UseLeftBorderScroller

NAME

Window.UseLeftBorderScroller – enable left border scrollbar

FUNCTION

If set to `True`, the window will feature a scrollbar in its left border. You must set this for the window object if any children are going to use this window border scroller, e.g. prop gadgets with the `Prop.UseWinBorder` attribute.

Obviously, scroll gadgets in window borders won't look good with borderless or non-resizable windows.

TYPE

Boolean

APPLICABILITY

I

53.31 Window.UseRightBorderScroller

NAME

Window.UseRightBorderScroller – enable right border scrollbar

FUNCTION

If set to `True`, the window will feature a scrollbar in its right border. You must set this for the window object if any children are going to use this window border scroller, e.g. prop gadgets with the `Prop.UseWinBorder` attribute.

Obviously, scroll gadgets in window borders won't look good with borderless or non-resizable windows.

TYPE

Boolean

APPLICABILITY

I

53.32 Window.Title

NAME

Window.Title – set title of window

FUNCTION

Specify the title of a window.

TYPE

String

APPLICABILITY

ISG

53.33 Window.Width

NAME

Window.Width – set/get window width

FUNCTION

Specify the width of a window. Usually, you won't give a pixel value here but instead use one of the following magic macros:

`Default`	Calculated from objects default sizes.
`MinMax:<0..100>`	Somewhere between the minimum width (0) and the maximum width (100) of your window.
`Visible:<1..100>`	Percentage of the screens visible width.
`Screen:<1..100>`	Percentage of the screens total width.
`Scaled`	Width will be adjusted so that width : height == minimum width : minimum height. Note that a windows width and height may not both be scaled.

Default for this tag is `Default`.

As long as your window has a window id (`Window.MuiID`), choosing a size is not that important. MUI will always remember a windows last position and size and these values will simply override your settings. Positioning and sizing should be completely under user control, a programmer doesn't need to worry about it.

TYPE

Number or predefined macro

APPLICABILITY

IG

Appendix A Licenses

A.1 MUI license

A.2 Expat license

A.3 LGPL license

library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted,

GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense,

Index