RapaGUI 2.1

Rapid Cross-Platform GUI Development On All Islands

Andreas Falkenhahn

Table of Contents

1 General information

1.1 Introduction

RapaGUI is a plugin for Hollywood that allows you to easily create GUIs with Hollywood. You just have to define the GUI layout in an XML file which is converted into a full-blown GUI by RapaGUI on the fly. It just doesn't get any easier!

RapaGUI's premium feature is definitely its cross-platform design. Like Hollywood, RapaGUI was designed as a platform-independent program. Therefore, it runs on a wide variety of platforms, but still, it uses OS-native widgets on all supported platforms to give your applications a truly authentic look and feel. Currently, RapaGUI is available in native versions for Windows, Linux (GTK2/GTK3), macOS, AmigaOS, MorphOS, AROS, and even for Android. With RapaGUI you just have to write your application once and it will automatically run on many other platforms too! This allows you to spend your time on the really important things, i.e. program design, instead of having to write backends for lots of different platforms.

RapaGUI uses an object-oriented design composed of over 40 MOAI (Magic Omnigui Architecture Interface) classes. Those MOAI classes constitute the heart of RapaGUI. All GUI elements supported by RapaGUI (windows, widgets, menu bars...) are simply objects derived from those MOAI classes. By wrapping the diverse native OS GUI APIs into platform-independent MOAI classes, those classes reduce the many faces of the different OS GUI APIs into just a single MOAI API face, carved in stone by RapaGUI!

RapaGUI supports all the widgets you need to create modern GUI applications, including multi-column listviews, treeviews, tabbed page widgets, toolbars, text editor widgets, menu bars, HTML views and much more. The highlight of RapaGUI, however, is certainly its inbuilt Hollywood MOAI class. This class allows dynamic embedding of complete Hollywood displays into GUIs which can be used to combine Hollywood's powerful multimedia functionality with RapaGUI's GUI abilities into one powerful application.

RapaGUI comes with extensive documentation in various formats like PDF, HTML, AmigaGuide, and CHM that describes the GUI programming basics in detail and provides a convenient MOAI function and class reference. A step-by-step tutorial that guides you through your first RapaGUI program is also included. On top of that, many example scripts are included in the distribution archive, including advanced scripts like a complete video player which really show off the power of Hollywood and RapaGUI working together.

All this makes RapaGUI the ultimate cross-platform GUI toolkit, carefully crafted for you sailors of the seven GUI seas! Only RapaGUI allows rapid cross-platform GUI development on all islands - it is the ultimate fusion of all the different OS GUI toolkits into one MOAI face, carved in stone for eternity and beyond.

1.2 Terms and conditions

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.

1.3 Requirements

RapaGUI requires at least Hollywood 9.0. Depending on the platform there are the following additional requirements:

Windows version:

– requires at least Windows XP

macOS version:

– Intel Macs: requires at least macOS 10.10
– PowerPC Macs: requires at least macOS 10.5

Linux version:

– requires GTK+ 2 or GTK+ 3, depending on which version of RapaGUI you use (for most architectures, RapaGUI is available in a build for GTK+ 2 and GTK+ 3)
– optional: WebKitGTK+ is needed for HTMLview class

AmigaOS version:

– requires MUI 3.8 or better; using MUI 4 (or better) is highly recommended, though!
– 68020+ or PowerPC processor
– CyberGraphX or Picasso96
– codesets.library for UTF-8 support
– Toolbar class requires TheBar.mcc
– Texteditor class requires TextEditor.mcc
– HTMLview class requires HTMLview.mcc

2 About RapaGUI

2.1 History

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

2.2 Compatibility notes

RapaGUI 2.0 API changes

Unfortunately, it was necessary to make a few design changes in RapaGUI 2.0. This might require some adaptations in your scripts to make them compatible with RapaGUI 2.0. Please read through the following notes to learn if your script is affected.

- Events are now always handled when they occur. For example, let's assume you have installed a notification on the `Radio.Active` attribute and then you do something like this:

  ```
  moai.Set("radio", "active", 5) ; triggers event callback!
  p_DoSomething()
  ```

  With RapaGUI 1.x, the event handler function for `Radio.Active` wouldn't be called before the next call to `WaitEvent()`, i.e. `p_DoSomething()` would always be called before the event handler for `Radio.Active` got called. In RapaGUI 2.0 this is different now. The event handler will always be called right when the event occurs. This means that `p_DoSomething()` will be called after the event handler for `Radio.Active` because the call to `moai.Set()` will force the event handler to be run immediately after changing the active radio item.
- `WaitEvent()` will never return. On most platforms, a call to `WaitEvent()` will be a one-way ticket that starts your application's main event loop and never returns. Only on AmigaOS and compatibles does `WaitEvent()` still return from time to time but you shouldn't rely on this behaviour because on other platforms it behaves differently. If you want `WaitEvent()` to run custom callbacks, just use Hollywood 9.0's new `RunCallback()` function.
- All RapaGUI projects must use `WaitEvent()` now. It's no longer acceptable to implement custom event handling using `CheckEvent()` or `CheckEvents()`. Routing normal Hollywood scripts through RapaGUI will only work if they use `WaitEvent()` too.
- `Dialog.ShowModal` no longer supports dialog functions. If you want to open a dialog that doesn't block your script, you now have to set `Window.Open` to `True` for the dialog (and possibly set `Application.Sleep` to `True` too) and then you can implement the dialog's actual behaviour by repeatedly calling a management function either using `SetTimeout()` or Hollywood 9.0's new `RunCallback()` function. Take a look at the updated Dialogs example for a reference implementation of a non-blocking dialog that uses this technique (choose `"Test progress bar dialog"` from the menu).
- Since modal event loops are no longer supported by RapaGUI, the plugin now disables all Hollywood functions which attempt to start a modal event loop, e.g. `WaitLeftMouse()`, `WaitSampleEnd()`, `InKeyStr()`, etc. These can't be used together with RapaGUI any longer.

- RapaGUI requires Hollywood 9 now. RapaGUI 2.0 uses many interfaces only available in Hollywood 9 so you cannot use it with older Hollywood versions any longer.
- When setting `Texteditor.Text`, RapaGUI will no longer trigger a `Texteditor.HasChanged` notification. Instead, `Texteditor.HasChanged` will just be set to `False` without triggering any event.
- `Toolbar.ViewMode` no longer defaults to `TextGfx` but to `Gfx`.
- When a listview has sortable columns, all items will always be sorted now. It is no longer possible to insert entries at arbitrary positions in a listview that has sortable columns. Instead, the entries are always sorted. Sorting is determined by the column that has the sort focus. See Section 27.40 [Listview.SortColumn], page 158, for details.

2.3 Future

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

- port RapaGUI to iOS
- support for Scintilla for an advanced cross-platform editing component
- support for advanced user interfaces using wxAUI (docking etc.)
- support for drag'n'drop
- support for more widgets

2.4 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. What am I doing wrong?

A: There are some things that you need to keep in mind for the resize feature to work correctly. If there is a widget 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. RapaGUI won't be able to resize this window unless you add rectangle objects to the sides of your image object because RapaGUI 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: Widgets of label class are actually not resizable because resizable labels look quite awkward!

Q: RapaGUI doesn't run on my Linux distribution although I have installed GTK+. What could be the reason for this?

A: First make sure that you are using the correct RapaGUI version. RapaGUI for Linux is available in versions for GTK+ 2 and GTK+ 3. Typically, Linux distributions only have one of the two GTK versions installed so you need to use a RapaGUI build that matches your install GTK version. You could also be missing some third party shared objects required by RapaGUI. To see what shared objects are required by RapaGUI, run the following command in a console: `readelf -d rapagui.hwp`

Q: When run on a high-DPI monitor my application appears blurry on Windows. Why?

A: Make sure to set the `DPIAware` tag in Hollywood's `@OPTIONS` preprocessor command to `True`. See Section 3.12 [High-DPI support], page 20, for details.

Q: RapaGUI's treeview widget doesn't work on Android. Why?

A: Android doesn't have a native treeview widget which is why RapaGUI can't support treeviews on Android.

2.5 Credits

RapaGUI was written by Andreas Falkenhahn. The design was inspired by my MUI Royale plugin which only runs on AmigaOS and compatibles and was first published at the end of 2012. First experiments with a wxWidgets-based GUI toolkit for Hollywood were already started in 2013. Initially I just planned to create a wrapper plugin that allows Hollywood scripts to use wxWidgets but then I realized that the wxWidgets API can be quite complicated to use and I thought that a MUI Royale-based approach would be much more user-friendly and convenient for GUI scripting. By imitating the MUI programming paradigms in RapaGUI I could also release versions for AmigaOS and compatibles making RapaGUI the first cross-platform GUI toolkit to support the Amiga's native MUI toolkit. Thus, RapaGUI has the potential to make a dream of many Amiga users come true: To have a cross-platform GUI toolkit which uses native widgets on all platforms! RapaGUI has finally accomplished this mission.

Thanks have to go to the wxWidgets team and Stefan Stuntz for their wonderful GUI toolkits. Special thanks go to Vadim Zeitlin and Eric Jensen for their valuable help on wxWidgets and Thore Böckelmann for his instant MUI fixes and his openness towards useful MUI extensions which he also implemented very quickly.

Additional thanks to Alfonso Ranieri for TheBar.mcc and codesets.library, Allan Odgaard for TextEditor.mcc, HTMLview.mcc and the TheBar.mcc, HTMLview.mcc and TextEditor.mcc Open Source Teams for maintaining those 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 Conceptual overview

3.1 Application tree

Every RapaGUI application is basically a tree containing a lot of MOAI objects that make up the application. Typically, MOAI objects are just widgets like buttons, listviews, combo boxes, etc. but they can also top-level windows or abstract objects like group objects which are used to compute the GUI layout.

The root element of every application tree must always be an instance of Application class. No MOAI object can exist outside the application object. Application objects are created by calling `moai.CreateApp()` and there can be only one application object in every program. The application object is responsible for handling all the events and messaging required by your application. See Section 9.1 [Application class], page 61, for details.

The most important children of the application object are the top-level windows of your application. These can be normal top-level windows or modal dialogs which block the rest of your application while they are open. To create those objects in XML, you just have to declare them using the `<window>` and `<dialog>` tags. See Section 62.1 [Window class], page 301, for details. See Section 16.1 [Dialog class], page 95, for details.

Every window or dialog always needs to have exactly one root object which must be derived from Group class. Group class allows you to combine one or more widgets in a horizontal, vertical, or grid-based layout. This is one of the most important classes and it can be accessed from XML by using the `<vgroup>`, `<hgroup>`, and `<colgroup>` tags. Whenever the window size changes, all groups are relayouted automatically which means that you don't have to care about the complex task of recalculating your GUI layout based on the current window size. RapaGUI does this all automatically for you. See Section 18.1 [Group class], page 105, for details.

Another very important class is Area class. All widgets are children of Area class because Area class basically just describes a rectangular region within the GUI layout on which widget-dependent graphics are drawn. This means that you can use all attributes and methods of this class on all MOAI objects that are widgets. For example, you can set the dimensions of all your widgets by just using the attributes `Area.Width` and `Area.Height`. This is usually unnecessary, however, because RapaGUI automatically chooses the dimensions for all widgets. If you're unhappy with RapaGUI's choice, you can override it by using those attributes, though. See Section 10.1 [Area class], page 67, for details.

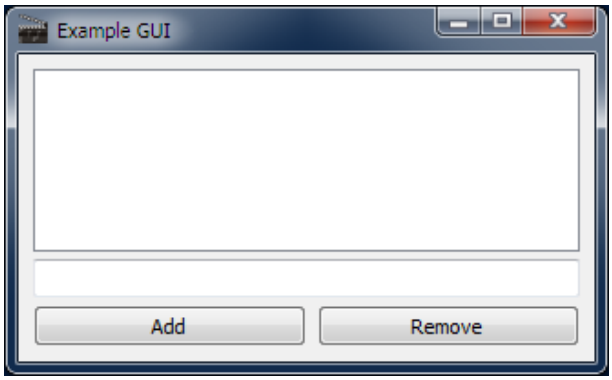
3.2 GUI layout

As you have already seen above, in RapaGUI GUIs are defined entirely using the XML markup language. This allows you to quickly create GUI layouts by just composing a tree hierarchy made up of several windows, groups, and widgets. Here is an example of what an application tree could look like in an XML declaration:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application>
   <window title="Example GUI">
      <vgroup>
         <listview>
```

```
            <column/>
         </listview>
         <textentry/>
         <hgroup>
            <button id="add">Add</button>
            <button id="rem">Remove</button>
         </hgroup>
      </vgroup>
   </window>
</application>
```

And here is what this GUI looks like on Windows 7:



As you can see in the XML, no position coordinates or sizes are ever given. This is all calculated automatically by RapaGUI to ensure a clean appearance on all systems. Keep in mind that RapaGUI applications can run on a wide variety of systems: On Windows, GTK, macOS, and even on AmigaOS. By omitting hard-coded window and widget sizes and positions RapaGUI can automatically choose the best sizes and positions depending on the current screen and font size. Of course, you can also force RapaGUI to use hard-coded sizes by using the `Area.Width` and `Area.Height` attributes. To enforce a listview size of at least 600x400 device-independent pixels, you could just write:

```
<listview width="600" height="400">
   <column/>
</listview>
```

The only thing which you cannot modify is the widget position. Since the widget position highly depends on the current UI font size it really doesn't make sense to allow hard-coded widget positions in a cross-platform GUI toolkit because there will be massive differences between the different operating systems. Thus, RapaGUI doesn't allow you to hard-code widget positions.

3.3 Running GUIs

All you have to do to create and run GUIs declared using XML is to write a few lines of Hollywood code. Let's assume you saved the example GUI XML definition from above into a file named `GUI.xml`, all you would have to do now to create and show this GUI from a Hollywood script is creating a Hollywood script with the following lines:

```
@REQUIRE "RapaGUI"
```

```
moai.CreateApp(FileToString("GUI.xml"))
Repeat
   WaitEvent
Forever
```

That's all! When you run the Hollywood script from above, your GUI will automatically be created and shown, no matter if your script is running on Windows, Linux, macOS, or AmigaOS.

3.4 Initializing RapaGUI

All you have to do to make your script use RapaGUI instead of Hollywood's inbuilt graphics engine is adding the following line to the top of your script:

```
@REQUIRE "rapagui"
```

Alternatively, if you are using Hollywood from a console, you can also start your script like this:

```
Hollywood test.hws -requireplugins rapagui
```

RapaGUI accepts the following arguments in its `@REQUIRE` call:

`ScaleGUI:`
: When running RapaGUI on a high-DPI monitor, all raster graphics will automatically be scaled to fit to the monitor's DPI settings. If you don't want that, set this tag to `False`. See Section 3.12 [High-DPI support], page 20, for details. If this tag is set to `True` (also the default), you can control whether or not bilinear interpolation should be used during scaling using the `InterpolateGUI` tag (see below). Defaults to `True`. (V2.0)

`InterpolateGUI:`
: If `ScaleGUI` is set to `True` (also the default), the `InterpolateGUI` tag can be used to specify whether or not bilinear interpolation should be used when scaling. Defaults to `True`. (V2.0)

`ScaleHollywood:`
: When running RapaGUI on a high-DPI monitor, all Hollywood widgets will automatically be scaled to fit to the monitor's DPI settings. If you don't want that, set this tag to `False`. See Section 3.12 [High-DPI support], page 20, for details. If this tag is set to `True` (also the default), you can control whether or not bilinear interpolation should be used during scaling using the `InterpolateHollywood` tag (see below). Defaults to `True`. (V2.0)

`InterpolateHollywood:`
: If `ScaleHollywood` is set to `True` (also the default), the `InterpolateHollywood` tag can be used to specify whether or not bilinear interpolation should be used when scaling. Note that in contrast to the `InterpolateGUI` tag the `InterpolateHollywood` tag defaults to `False` which means that by default, bilinear interpolation is not used for Hollywood widgets. (V2.0)

`HideDisplays:`
: By default, RapaGUI will hide all Hollywood displays on startup. If you don't want that, set this tag to `True`. This can be useful when using RapaGUI in

Hollywood emulation mode, i.e. without using any of RapaGUI's GUI functionality but just to get some features that Hollywood does not support by default on some platforms (e.g. menus on Linux, or inter-process communication on Linux and macOS). (V2.0)

Here is an example of how to pass arguments to the `@REQUIRE` preprocessor command:

```
@REQUIRE "rapagui", {ScaleGUI = False}
```

Alternatively, you can also use the `-requiretags` console argument to pass these arguments. See the Hollywood manual for more information.

3.5 Object handling

As you've already seen RapaGUI uses XML files to create MOAI 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 textentry object with a maximum length of 80 characters you would simply use the `TextEntry.MaxLen` attribute and include it in your XML declaration.

```
<textentry id="mystring" maxlen="80"/>
```

Once your object is ready, you can start talking to it by setting or getting its attributes or by running its methods. For that purpose it is important that you give your object an identifier using the `id` attribute so that the functions `moai.Set()`, `moai.Get()` and `moai.DoMethod()` can find your object. In the code above we assigned the id `mystring` to our textentry widget. Here's an example of how we could talk to this widget now:

```
moai.Set("mystring", "text", "look")
s$ = moai.Get("mystring", "text")
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.

The next thing you have to do is handling events that are triggered by your GUI. This is covered in the next section.

3.6 Event handling

RapaGUI events are handled in the very same way as normal Hollywood events. This means that you just have to pass a callback function to Hollywood's `InstallEventHandler()` function and whenever a RapaGUI event is triggered, your callback function will be called.

Here is an example how to install a RapaGUI event callback:

```
InstallEventHandler({RapaGUI = p_EventHandler})
```

Whenever an event occurs, RapaGUI will then call `p_EventHandler()` with a table as parameter. The table will have the following fields initialized:

`Action:` Initialized to "RapaGUI".

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

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

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

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

`MOAIUserData:`
If the object was assigned certain userdata, it will be passed to the event handler callback in this tag. See Section 33.8 [MOAI.UserData], page 187, 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 33.7 [MOAI.NotifyData], page 187, for details.

On top of that it is also necessary to tell RapaGUI which events you would like to receive. Only events you explicitly request to receive are passed to your event callback to minimize overhead. An exception are button, toolbar button and menu item events. They are passed to your event callback even if you haven't requested them. The reason for this design choice is that normally all of these events need to be individually handled because the user expects something to happen when pressing a button.

All RapaGUI events are coupled to attributes of MOAI objects. The next section describes this in detail.

3.7 Attribute notifications

RapaGUI's event handling is based on the values of certain attributes of the different MOAI classes. You can listen to the values of all attributes that have an applicability of `N`. If you set up a listener on an attribute value, RapaGUI will run your event callback whenever the attribute value changes.

For example, you could set up a listener on the `Listview.Active` attribute which contains the active item of a listview widget. If you do this, your event handler will be called whenever the active item of the respective listview widget changes. To install a listener on a certain attribute, you have to set up a notification for this attribute in your object declaration. This is done directly in the XML file by using the `Notify` attribute that is accepted by all MOAI classes:

```
<listview id="lv" notify="active">
   <column>
      <item>One</item>
      <item>Two</item>
      <item>Three</item>
   </column>
</listview>
```

The code above will run your event callback whenever the active item of the listview object that has the id `lv` changes because you've requested to listen to the value of the `Listview.Active` attribute.

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

```
<listview id="lv" notify="active; doubleclick">
...
</listview>
```

As you can see, the code above installs listeners on both attributes, `Listview.Active` and `Listview.DoubleClick`, so that your event callback is also invoked when the user double clicks on a listview entry.

Alternatively, you can also set up or remove notifications at run-time using the `moai.Notify()` function from your code.

To print the currently active listview item from your event handler, you could then use the following code:

```
Function p_EventHandler(msg)
    Switch msg.action
    Case "RapaGUI":
       Switch msg.attribute
       Case "Active":
          Switch msg.id
          Case "lv":
             DebugPrint("Active listview item:", msg.triggervalue)
          EndSwitch
       EndSwitch
    EndSwitch
EndFunction
```

Keep in mind, though, that attribute values can also be changed manually. For example, your program might want to manually activate a certain listview item by doing something like this:

```
moai.Set("lv", "active", 5)  ; activate item number 6
```

Since the call above changes the value of the `Listview.Active` attribute as well, your event handler callback will be triggered by this call, too. If you don't want this, you can use the special attribute `MOAI.NoNotify` in your call to `moai.Set()`. Whenever `MOAI.NoNotify` is set to `True`, the attribute's value will be changed without invoking any event handlers. Thus, to change the active listview item without triggering any event callbacks, you would just write:

```
moai.Set("lv", "active", 5, "nonotify", True)
```

Finally, as already mentioned in the previous section, there are some attributes that RapaGUI always listens to. These are: `Button.Pressed`, `Button.Selected`, `Toolbarbutton.Pressed`, `Toolbarbutton.Selected`, and `Menuitem.Selected`. Since these are so common and widely used and you will normally always want to listen to attributes like `Button.Pressed` because pressing a button will always cause a reaction, RapaGUI listens to them automatically. Thus, you don't have to explicitly request a notification on these attributes, i.e. writing the following is redundant:

```
<button id="btn" notify="pressed">Click me</button>
```

Instead, you can just write:

```
<button id="btn">Click me</button>
```

The same is true for menu items and toolbar buttons.

3.8 Dynamic objects

Most applications create their GUI by a simple call to `moai.CreateApp()` which is passed an XML declaration that contains the complete user interface of the whole application. In some situations, though, it might be necessary to create MOAI objects after the call to `moai.CreateApp()` and insert them into the existing application.

This is possible by using the `moai.CreateObject()` API. This function allows you to create a MOAI object from an XML declaration, pretty similar to what `moai.CreateApp()` does except that `moai.CreateObject()` doesn't allow you to create an application because there can be only one application.

For example, you can create a button using `moai.CreateObject()` like this:

```
moai.CreateObject([[<button id="mybutton">Click me</button>]])
```

It is very important to set an identifer for the MOAI object you create using `moai.CreateObject()` because the identifier is needed to refer to the object later. Now that we have created the button we could insert it into an existing window layout. Let's suppose our window layout currently looks like this:

```
<window>
   <hgroup id="mygroup">
      <button id="ok">OK</button>
      <button id="cancel">Cancel</button>
   </hgroup>
</window>
```

We could now use the `Group.Insert` method to insert the newly created button after the "OK" button. We would have to use the following code to do that:

```
moai.DoMethod("mygroup", "initchange")
moai.DoMethod("mygroup", "insert", "mybutton", "ok")
moai.DoMethod("mygroup", "exitchange")
```

You can see that we're also calling the methods `Group.InitChange` and `Group.ExitChange`. This is very important. Those methods have to be run whenever you change the children of a group, i.e. you remove or add children. RapaGUI needs you to call these methods because after removing or adding group children a window relayout becomes necessary. That's why simply calling `Group.Insert` or any other method that changes the number of group children isn't enough.

Of course, we can also remove children from groups. This is possible by using `Group.Remove`. The following code removes the "Cancel" button from the above window layout:

```
moai.DoMethod("mygroup", "initchange")
moai.DoMethod("mygroup", "remove", "cancel")
moai.DoMethod("mygroup", "exitchange")
```

After this call, the MOAI object with the id "cancel" has become a detached object. This means that you could also attach it to a group now again by using `Group.Insert` or a similar method. Note that all methods which insert MOAI objects into existing layouts only accept detached objects. It's not possible to insert the same MOAI object into multiple groups. As soon as you insert a MOAI object into a group (or menu tree), it changes its state from detached to attached. Once a MOAI object is in attached state, it can no longer be passed to methods which expect a detached object. In order to turn a MOAI object from attached

to detached state, you need to use a method like `Group.Remove` to detach a MOAI object from a group.

As a final note, the XML code you pass to `moai.CreateObject()` can also be a complete tree, e.g. you could create a complete window with just a single call to `moai.CreateObject()`. This is all possible. `moai.CreateObject()` isn't limited to creating single MOAI objects but it can create multiple objects for you as well. When you're creating a window or dialog with `moai.CreateObject()`, though, don't forget to add it to the application object by calling `Application.AddWindow` since window objects are also in detached state by default unless they have been added to an application object. For dialogs, you can just use `moai.CreateDialog()` which will implicitly call `Application.AddWindow`.

3.9 Sizeability

One of RapaGUI's premium features is its capability to automatically recalculate the complete GUI layout when resizing the window that contains the widgets. This, however, is only possible if all groups in a window can be resized, otherwise the window's size will stay fixed and won't be resizable. In order to create windows that are freely resizable, you need to know which widgets are resizable and which are not. Consider the following GUI definition:

```
<window>
   <vgroup>
      <button id="btn">Hello World!</button>
   </vgroup>
</window>
```

This window will only be horizontally resizable because button widgets are only horizontally resizable by default. They aren't vertically sizable by default because buttons which change their vertical size look pretty ugly. Normally, a button's height is fixed to the standard button height as defined by the theme currently active on the operating system.

If you want this window to be resizable, you have several options: A widely used option is to insert empty space using objects of Rectangle class. These are resizable in all directions. Thus, if you insert a `<rectangle>` object, your window will suddenly become resizable in all directions:

```
<window>
   <vgroup>
      <button id="btn">Hello World!</button>
      <rectangle/>
   </vgroup>
</window>
```

Alternatively, you could also insert a different widget which is vertically resizeable, for example a listview. Or you could even make the button vertically resizable by setting the `Area.FixHeight` attribute to `False`:

```
<window>
   <vgroup>
      <button id="btn" fixheight="false">Hello World!</button>
   </vgroup>
```

```
</window>
```

But this doesn't look so pretty because now the user could end up with a button that is a few hundred pixels in height. That's why the `<rectangle>` approach is the one most oftenly used to insert padding space for non-resizable objects.

In order to use `<rectangle>` objects as padding space in the right positions, you need to know which widgets are resizeable by default and which are not. Therefore, here is an overview of the sizeability of the individual widgets supported by RapaGUI:

`Busybar class:`
: Horizontally resizable.

`Button class:`
: Horizontally resizable.

`Checkbox class:`
: Not resizable.

`Choice class:`
: Horizontally resizable.

`Combobox class:`
: Horizontally resizable.

`Hollywood class:`
: Not resizable.

`HLine class:`
: Horizontally resizable.

`HSpace class:`
: Vertically resizable.

`HTMLview class:`
: Resizable in all directions.

`Image class:`
: Not resizable.

`Label class:`
: Not resizable.

`Listview class:`
: Resizable in all directions.

`Pageview class:`
: Resizable in all directions.

`Popcolor class:`
: Horizontally resizable.

`Popfile class:`
: Horizontally resizable.

`Popfont class:`
: Horizontally resizable.

`Poppath class:`
: Horizontally resizable.

`Progressbar class:`
: Horizontally resizable for horizontal progress bars. Vertically resizable for vertical progress bars.

`Radio class:`
: Not resizable.

`Rectangle class:`
: Resizable in all directions.

`Scrollbar class:`
: Horizontally resizable for horizontal scrollbars. Vertically resizable for vertical scrollbars.

`Scrollcanvas class:`
: Resizable in all directions.

`Scrollgroup class:`
: Resizable in all directions.

`Slider class:`
: Horizontally resizable for horizontal sliders. Vertically resizable for vertical sliders.

`Text class:`
: Horizontally resizable.

`Texteditor class:`
: Resizable in all directions.

`Textentry class:`
: Horizontally resizable.

`Textview class:`
: Resizable in all directions.

`Treeview class:`
: Resizable in all directions.

`VLine class:`
: Vertically resizable.

`VSpace class:`
: Horizontally resizable.

Of course, you can change these defaults by setting the `Area.FixWidth` and `Area.FixHeight` attributes accordingly to enable or disable certain sizeability settings, but this is often not recommended because it will look pretty ugly if for example a textentry widget is suddenly vertically resizable. This would unnecessarily confuse the user because he might end up with a textentry widget which is 300 pixels in height but only allows to enter one line of text. Thus, the best idea is to respect the OS defaults for the standard widgets.

Of course, there are also widgets where it's completely acceptable to override the defaults from above. For example, widgets deriving from Hollywood class are non-resizable by default but of course you are encouraged to use `Area.FixWidth` and `Area.FixHeight` to adjust the widget's sizeability to your personal needs.

3.10 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 `moai.Set()` at runtime.

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

`N` Notifications on this attribute are possible either by using the "Notify" attribute in the XML declaration or by calling the `moai.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 `moai.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.

3.11 Device-independent pixels

All position and size specifications in RapaGUI must be in device-independent pixels. This has the advantage that they will be independent of the current monitor's DPI setting so that your application will still look correctly when run on high-DPI monitors, for example.

Consider the following declaration of a window:

```
<window width="640" height="480">
   ...
</window>
```

This will declare a window whose width is 640dip and whose height is 480dip. If the current monitor doesn't use DPI scaling (e.g. 96dpi on Windows, 72dpi on macOS), the device-independent pixels will be identical to physical pixels so that the window's size will be 640x480 pixels. If the monitor is configured to use DPI scaling, however, the device-independent pixels will automatically be adapted to the monitor's DPI setting. For example, on a 192dpi monitor (i.e. 200% scale on Windows) the window's physical dimensions on the screen will be 1280x960 pixels instead because RapaGUI will automatically adapt the device-independent pixels to the current monitor's DPI.

Note that it is generally a good idea to avoid hard-coding specific positions and sizes and let RapaGUI determine these automatically from your layout. This guarantees a maximum amount of compatibility between different operating systems and their specific window managers, fonts, themes, and decorations.

Also make sure to read the chapter on High-DPI support to learn about other things to consider when designing an application that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

3.12 High-DPI support

RapaGUI has full support for high-DPI displays. Normally, RapaGUI handles everything automatically for you so that your application should look the same regardless of the monitor's DPI setting. This is achieved by handling all position and size values in device-independent pixels which makes RapaGUI applications scale nicely to different monitor DPI settings. Furthermore, RapaGUI will automatically scale all raster graphics, e.g. button images set using `Button.Icon`, to fit to the monitor's DPI setting.

Note that RapaGUI's automatic scaling of raster graphics could lead to images becoming blurry on high-DPI monitors. You can change that by either providing sets of images for different DPI settings instead of just a single image or by using vector images that can be scaled with no losses in quality to any size.

For example, let's suppose you want to add a 32x32px icon to your button using the `Button.Icon` attribute. If you set `Button.Icon` to a raster brush and the user's monitor has a DPI scaling of 200% activated, RapaGUI will scale the image to 64x64px which will look blurry. So you could either set `Button.Icon` to the identifier of a vector brush or you could create a Hollywood icon, add 32x32px and 64x64px images (and possibly more) to it, and then pass this icon to `Button.Icon`.

Note that in that case you'll also have to set the `Button.IconType` attribute to `Icon` to tell RapaGUI that you've passed the identifier of an icon instead of a brush, which is the default type for `Button.Icon`. In that case, RapaGUI will use the 32x32px image inside the icon in case the monitor has no DPI scaling activated or it will use the 64x64px image in case the monitor has a DPI scaling of 200% activated. If the DPI scaling is something between 100% and 200%, RapaGUI will scale the image inside the icon whose size is closest to the desired target size.

By default, all widgets that support images expect you to pass Hollywood brushes. If you always want to pass Hollywood icons instead, you can also set the global attribute `Application.UseIcons` to `True`. This will change the default of all tags accepting images from Hollywood brushes to Hollywood icons.

As described above, when providing images to RapaGUI using Hollywood brushes that contain raster data, RapaGUI will automatically scale those raster brushes to fit to the current monitor's DPI settings. If you don't want that, you can set the `ScaleGUI` tag to `False` when you `@REQUIRE` the RapaGUI plugin. See Section 3.4 [Initializing RapaGUI], page 11, for details. Bilinear interpolation during scaling can be turned off by setting the `InterpolateGUI` tag to `False` on `@REQUIRE`. Alternatively, automatic image scaling can also be configured on a per-widget basis. For example, for button images you can control automatic image scaling by setting the `Button.IconScale` tag.

Also, all Hollywood widgets will be automatically scaled to fit to the current monitor's DPI settings. If you don't want that, you can set the `ScaleHollywood` tag to `False` when you `@REQUIRE` the RapaGUI plugin. See Section 3.4 [Initializing RapaGUI], page 11, for details. Bilinear interpolation during scaling can be enabled by setting the `InterpolateHollywood` tag to `True` on `@REQUIRE`.

With Scrollcanvas class the dimensions you specify in `Scrollcanvas.VirtWidth` and `Scrollcanvas.VirtHeight` are in device-independent pixels by default and RapaGUI will automatically apply the system's scale factor to the contents drawn by the `Scrollcanvas.Paint` function. If you want to have fine-tuned control, you can set the `Scrollcanvas.AutoScale` attribute to `False`. In that case, `Scrollcanvas.VirtWidth` and `Scrollcanvas.VirtHeight` are interpreted as physical pixels and no auto scaling will be done so that your paint function can draw high resolutions graphics without any quality loss due to scaling.

Finally, keep in mind that you explicitly have to enable DPI-aware mode on Windows if you want your application to support high-DPI modes natively. This can be done by setting the `DPIAware` tag in the `@OPTIONS` preprocessor command to `True`, e.g.

```
@OPTIONS {DPIAware = True}
```

If you don't do that, your application won't be DPI-aware which means that Windows will run it in a special compatibility mode. In this compatibility mode, Windows will scale your application to match the monitor's DPI setting but this will lead to a blurry appearance. That's why it is suggested to set `DPIAware` to `True`, as shown above.

Note that on Linux you should use the GTK+ 3 version of RapaGUI if you want to support high-DPI modes because GTK+ 2 doesn't support high-DPI modes very well.

3.13 Keyboard shortcuts

RapaGUI allows you to set up keyboard shortcuts really easily by simply using an underscore character (`"_"`) before the character you would like to set up as a keyboard shortcut. You should always provide support for keyboard shortcuts because many users prefer to use them instead of the mouse, especially when it comes to actions which have to be repeated dozens of times it is much easier to use the keyboard instead of, just as an instance, having to navigate to certain submenus hidden deep within the menu hierarchy all the time. The following classes support keyboard shortcuts defined through the underscore character:

- Button class
- Checkbox class
- Label class
- Menu class
- Menuitem class
- Radio class
- Slider class
- Text class
- Toolbarbutton class

Here is an example of how to set up two keyboard shortcuts for widgets derived from Button class:

```
<hgroup>
   <button id="ok">_OK</button>
   <button id="cancel">_Cancel</button>
</hgroup>
```

The XML code above will set up "O" and "C" as keyboard shortcuts. On Windows, the user has to press `ALT+O` to select the "OK" button and `ALT+C` to select the "Cancel" button. On other platforms the combinations are sometimes different, e.g. on macOS you have to use the `COMMAND` key instead of the `ALT` key.

Since Label class doesn't create widgets that can be controlled by the user, specifying a keyboard shortcut for labels simply activates the next widget in the GUI layout, which is usually the widget that is described by the label. For example:

```
<hgroup>
   <label>_Name</label>
   <textentry/>
</hgroup>
```

As you can see, in the XML code above "N" has been set up as a keyboard shortcut by using an underscore character. Pressing `ALT+N` will activate the text entry widget then since label widgets cannot take the window focus.

In the rare case that you want to use the underscore character as a label text for one of your widgets and not have it automatically converted into a token indicating a keyboard shortcut, you can just set the `Area.NoAutoKey` attribute to `True`. If this is set, RapaGUI will simply show the underscore character and won't treat it as a special token. Note: Since menu and menuitem objects aren't derived from Area class, you need to use `Menu.NoAutoKey` and `Menuitem.NoAutoKey` in case you want to disable automatic shortcut generation for menu or menuitem objects.

If you need to use more complex keyboard shortcuts (e.g. certain function or control keys, combinations of keys), you have to set up an accelerator table using Accelerator class for your window. Accelerator class allows the definition of advanced keyboard shortcuts which are also independent of any widgets in your window. If your complex keyboard shortcuts are linked to menu items, however, you don't have to use Accelerator class but you can just set the `Menuitem.Shortcut` attribute to the desired shortcut. This will automatically set up an accelerator then. See Section 32.6 [Menuitem.Shortcut], page 180, for details. See Section 7.1 [Accelerator class], page 57, for details.

3.14 Text formatting codes

Widgets of type Textview class and Texteditor class support text formatting if the `Styled` attribute of those classes is set to `True`. RapaGUI supports text formatting by special control codes which are described in this section.

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:

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

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:

```
moai.Set("mytext", "text", "\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 text style to underline.

`\33b` Set the text style to bold.

`\33i` Set the text style to italic.

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

`\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. On AmigaOS and compatibles this requires MUI 4.0.

3.15 Implementing help texts

RapaGUI supports several ways of implementing context-sensitive help texts in your applications. First of all, every widget can have tooltips that pop up after the mouse has been hovering over a widget for some time. This is done via the `Area.Tooltip` attribute. As Area class is the super class for all widgets, you can use this attribute to add tooltip help to all your widgets. Here is an example:

```
<vgroup>
   <listview tooltip="List of loaded video files">
      <column/>
   </listview>
   <button id="btn1" tooltip="Plays a video stream">Play</button>
   <button id="btn2" tooltip="Stops a video stream">Stop</button>
   <button id="btn3" tooltip="Exits the program.">Quit</button>
</vgroup>
```

Furthermore, menu items and toolbar buttons support the attributes `Menuitem.Help` and `Toolbarbutton.Help`, respectively. If your window has a status bar attached, then the text you specify in these attributes is automatically shown in the status bar whenever the mouse cursor is over the respective menu item or toolbar button. This kind of visual feedback is very helpful because it is shown immediately in the status bar and the user doesn't have to wait for a few seconds as it is the case with tooltips.

3.16 Context menus

RapaGUI supports context menus for each of its widget classes. Context menus are assigned to the individual widgets by using the `Area.ContextMenu` attribute. As Area class is the super-class for all widgets, you can use this attribute with every MOAI object that creates a widget. If you've declared `Area.ContextMenu` for a widget, RapaGUI will automatically show the context menu whenever the user clicks the right mouse button while the cursor is over a widget that has a context menu attached.

`Area.ContextMenu` expects an object derived from Menu class as its argument so you have to create such a menu MOAI object for your context menu in XML first. It is very important to note that you have to declare your menus in the `<application>` scope because menus are global objects and are only attached to windows or widgets later on. That is why it is not allowed to declare menus 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:

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

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

Note that when using Menu class to create context menus, the `Menu.Title` attribute is only used on AmigaOS and compatibles. Context menus on Windows, Linux, and macOS don't show a title.

Also note that context menu events will be delivered via the standard `Menuitem.Selected` mechanism and not through a special context menu event handler. Since you can use the same menu object as a context menu for several widgets, you need a way to find out the widget whose context menu triggered the event. To give you this information, the event message will contain an additional item named `"Parent"` which contains the ID of the context menu's hosting widget. This allows you to re-use the same menu object as a context menu for multiple parent widgets.

AmigaOS users please note that MUI doesn't allow context menu objects to be shared across windows. You are not allowed to use the same menu object with widgets in different windows. It's okay to re-use a menu object for several widgets in the same window but not for widgets in another window. All other platforms don't have this limitation, only AmigaOS is affected here.

3.17 Internationalization

RapaGUI easily allows you to add internationalization (i18n) support to your applications because it seamlessly integrates with Hollywood's catalog system. Thus, all you have to

do to support multiple languages in your application is create a catalog containing all language-dependent texts used by your program and open it at startup, for example using the `@CATALOG` preprocessor command, e.g.

```
@CATALOG "MyApp.catalog"
```

You can then address individual catalog entries by using either the `MOAI.I18N` attribute or the `@i18n:` directive. All attributes that accept a string argument also accept the `MOAI.I18N` attribute which allows you to specify a catalog string index that should be used if a catalog for the user's system language is available. The string index can either be an absolute numeric value or a Hollywood constant. For example, to create a multi-lingual button you could do the following:

```
<button i18n="0">Click me</button>
```

With such XML code, the text "Click me" will only be used if the user's system language is English or RapaGUI can't find a catalog for the current system language. Otherwise, RapaGUI will use the string at index 0 in the catalog file instead of the default text "Click me".

For for locale-dependent strings that are passed in tag attributes you have to use the `@i18n:` directive. Just append this suffix to the string, followed by a numeric value or a Hollywood constant and Hollywood will use the specified catalog string instead if there's a catalog for the user's system language available. For example:

```
<window title="My application@i18n:1">...</window>
```

In that example, RapaGUI will use the window title "My application" only if the user's system language is English or if RapaGUI can't find a catalog for the current system language. Otherwise, RapaGUI will use the string at index 1 in the catalog file instead of the title "My application".

As already pointed out, you can also use Hollywood constants instead of hard-coded numeric values. Using Hollywood constants might be more convenient to maintain your XML because it allows you to easily add and remove entries. Using Hollywood constants, the XML from above looks like this:

```
<button i18n="#CAT_BUTTON">Click me</button>
<window title="My application@i18n:#CAT_TITLE">...</window>
```

Note that all these internationalization features only apply to the XML. All changes that are made at runtime need to be handled manually, so you need to use Hollywood's `GetCatalogString()` function to obtain the correct catalog string when making locale-dependent changes to the GUI at runtime, e.g. by calling functions like `moai.Set()`.

3.18 Character encoding

RapaGUI 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 on AmigaOS and compatibles, RapaGUI will require `codesets.library` to do character conversion from UTF-8 to the system's default charset.

Amiga users please note that MUI does not support UTF-8. MUI always uses the system's default charset. Thus, if you use UTF-8 encoding in your XML files, RapaGUI will try to map these strings to the system's default charset. This must 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.

3.19 Hollywood bridge

A powerful feature of RapaGUI is that it allows you to embed complete Hollywood displays inside your 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 widget 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 GUI widget 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 widget. For this, the Hollywood display's size is changed to 320x200 and then it is embedded in the GUI using a MOAI object of type Hollywood class. Here is the XML GUI declaration first:

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

Note that we use MOAI 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 your GUI widget and Hollywood:

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

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 "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({RapaGUI = p_EventFunc})
moai.CreateApp(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:

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

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 "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({RapaGUI = p_EventFunc})
moai.CreateApp(FileToString("GUI.xml"))

Repeat
   WaitEvent
Forever
```

Finally, it is also possible to make the Hollywood object resizable by setting the attributes `Area.FixWidth` and `Area.FixHeight` to `False`. Whenever the user resizes the 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>
   <window title="Hollywood bridge">
      <vgroup>
         <hollywood display="1" fixwidth="false" fixheight="false"/>
         <hgroup>
            <button id="play">Play</button>
            <button id="stop">Stop</button>
         </hgroup>
      </vgroup>
```

```
   </window>
</application>
```

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:

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

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 "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({RapaGUI = p_EventFunc, SizeWindow = p_EventFunc})
moai.CreateApp(FileToString("GUI.xml"))

Repeat
   WaitEvent
Forever
```

From this example you can see that Hollywood class is really powerful and can be used to achieve lots of innovative GUI ideas only limited by your creativity!

3.20 Image cache

Many MOAI classes allow you to use icons with the widgets they create, e.g. you can add an icon to a button widget by simply using the `Button.Icon` attribute. All MOAI classes that can deal with icons require you to pass either a Hollywood brush or a Hollywood icon that shall be used as the icon to the class.

What is important to know now is that RapaGUI caches these images for reasons of performance and economy. Just imagine a listview with thousands of rows and an icon in each of them. It would be an absolute performance killer if RapaGUI had to convert these icons from a Hollywood brush/icon into a RapaGUI image for every single row. That's why the images are cached.

Take a look at this example:

```
<button id="ok" icon="1">OK</button>
```

The declaration above puts Hollywood brush number 1 next to the text `"OK"` on the button's label. RapaGUI will now cache the image data of Hollywood brush number 1 internally and whenever there is a reference to Hollywood brush number 1 again, it will simply take the copy from its cache instead of the original one. This means that any changes you make to Hollywood brush number 1 after the declaration above won't have any effect on RapaGUI at all because of its image cache! Please keep that in mind. You either have to use a brush identifier that is not in RapaGUI's image cache yet or you have to remove the brush from RapaGUI's image cache using the `moai.FreeImage()` function first. For listview and treeview widgets, it's also possible to use `moai.UpdateImage()` instead.

There is just one exception to the rule: Widgets derived from Image class don't cache any image data, although they deal with Hollywood brushes/icons as well. The reason for this

is that image widgets often display pretty large images that could also be animated or changed many times. That's why it doesn't make sense to cache image data for widgets derived from Image class. Images used in other widgets, e.g. buttons, toolbar buttons, listviews, treeviews, pageviews, are usually very small and there is usually just a fixed number of them. On top of that, they might be needed very often - just think of a listview with thousands of rows and an icon in each of those rows - which is why an image cache is really necessary here to guarantee a good performance.

3.21 Platform-dependent features

Generally speaking, RapaGUI tries to include as few platform-dependent features as possible to allow the same application to run on a wide variety of platforms without any adjustments. There are, however, very few attributes and methods that, although not available on all platforms, are still included because they were considered so important that they have been included although they somewhat contradict RapaGUI's platform-blind GUI toolkit approach. For example, the `Window.HideFromTaskBar` attribute is only available on Windows and GTK+ since there is no taskbar on AmigaOS and macOS. Another example is `Application.OpenConfigWindow` which is only available on AmigaOS and compatibles.

Features which are platform-specific are always documented as such. You just need to look into the documentation to see if a certain feature is platform-specific. To minimize platform-specific code, all platform-dependent features can be specified on all platforms supported by RapaGUI as well. RapaGUI will just ignore the attributes and methods then instead of reporting an error message. Thus, you could also set `Window.HideFromTaskBar` on AmigaOS and macOS. RapaGUI won't report an error message. The call will simply be ignored.

3.22 MUI Royale compatibility

RapaGUI started as a fork of the popular MUI Royale plugin for Hollywood and people familiar with MUI Royale will no doubt see that RapaGUI shares the very same design so porting applications from MUI Royale to RapaGUI isn't particularly difficult. Although there have been many semantic changes to abstract RapaGUI from AmigaOS-based terminology, the real design differences are actually very few. Thus, when porting applications from MUI Royale to RapaGUI most of the time you will just be busy with changing names. Most people probably won't even encounter one of the very few design differences between MUI Royale and RapaGUI. Still, here is a non-exhaustive list of design differences between MUI Royale and RapaGUI to simplify your porting endeavours:

- MUI Royale required you to handle `Window.CloseRequest` for all your windows or they would stay open. RapaGUI automatically closes windows that haven't got a listener for `Window.CloseRequest` installed.
- MUI Royale also required you to handle the `"HideWindow"` and `"ShowWindow"` events to allow the minimizing of windows. RapaGUI handles this automatically now.
- RapaGUI automatically sets up event handlers for buttons, toolbar buttons and menu items. MUI Royale required you to explicitly request them but this led to unnecessarily wordy XML files where you had to include something like `notify="pressed"` for each and every button because under normal circumstances you want to be notified about

each button click. RapaGUI automatically calls your event handler for all button and toolbar button clicks and menu item events now to make the XML more readable.

- RapaGUI automatically hides all Hollywood displays on startup whereas MUI Royale doesn't do this. On MUI Royale you manually have to declare the default display as hidden.
- MUI Royale allows you to use text formatting codes almost everywhere. Since this is a highly MUI-centric feature, this is impossible to implement in a platform independent way. RapaGUI supports text formatting only in its Textview and Texteditor classes and only if `Textview.Styled` and `Texteditor.Styled` have been set to `True`.
- MUI Royale also allowed you to include icons in almost all widgets via text formatting codes. This is also impossible to implement in a cross-platform way. Still, RapaGUI has icon support in many classes but the way this is implemented is always class dependent. On RapaGUI you can use icons with the following classes: Button class, Listview class, Pageview class, and Treeview class. Check out the documentation of these classes to learn more about how to use icons with them.
- MUI Royale requires you to use an object derived from `<virtgroup>` class as the child for `<scrollgroup>` objects. On RapaGUI you can just use normal groups.
- MUI Royale sends context menu events using the `Area.ContextMenuTrigger` attribute. RapaGUI simply sends context menu events as normal menu item events but it includes a `Parent` field in the event message to inform you about the widget whose context menu triggered the event.
- RapaGUI's Checkbox class class creates a checkbox and a text label. MUI Royale's checkmark class simply creates a checkbox image and you have to create the label yourself.
- RapaGUI doesn't have support for the `Listview.InsertPosition` attribute but its `Listview.Insert` method simply returns the position of the newly inserted entry.
- In MUI Royale text editor ranges were indicated by block coordinates consisting of four different values (x1, y1, x2, y2). RapaGUI just uses start and stop coordinates now, so all the methods and attributes that deal with text ranges just require/return two arguments now.
- RapaGUI's Texteditor class, Textview class, and Textentry class use the content between the opening and closing XML tags as the initial widget contents whereas MUI Royale always required you to use an attribute to set the initial contents.
- RapaGUI requires you to specify a parent object most of the times when creating objects dynamically using `moai.CreateObject()`. This wasn't necessary in MUI Royale.
- Since RapaGUI's Treeview class supports multiple columns, you have to create at least one `<column>` when declaring your treeview in XML. This wasn't needed with MUI Royale because its treeview class merely supports single column trees.
- MUI Royale doesn't have the concept of dialogs since AmigaOS doesn't have it either. Modal dialogs were just emulated using normal windows and putting all other windows to sleep. RapaGUI now introduces real dialogs. While with MUI Royale you would typically create all those pseudo-dialogs at startup, this is no longer recommended with RapaGUI. Since windows are quite a finite resource on some operating systems, you should create dialogs only when you need them and destroy them immediately

afterwards. See Section 16.1 [Dialog class], page 95, for details. If you only target AmigaOS, then you can of course still create all your windows with just a single call to `moai.CreateApp()` but if you want to target other operating systems as well, you should follow the recommended programming guidelines and create dialogs when you need them and destroy them as soon as you are finished with them.

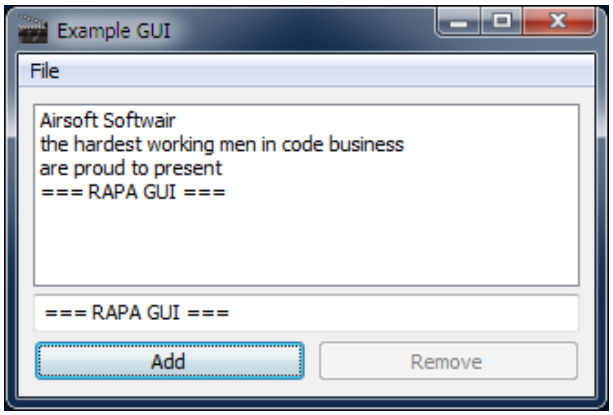
- Objects of Radio class are always framed in RapaGUI. In MUI Royale they aren't framed by default.
- When using a somewhat more complex shortcut string like `CTRL+V`, `Alt+X` or `F5` with `Menuitem.CommandString`, MUI Royale doesn't automatically listen to it. This has to be implemented manually. Only standard shortcuts in the form of a single alphabetical key combined with the `CMD` key are handled automatically by MUI Royale. RapaGUI, however, automatically handles complex shortcuts as well.
- When getting `Listview.Active` and no entry is active, MUI Royale returns the special string `Off` whereas RapaGUI returns -1.

Of course, there are also many attributes and methods that are present in MUI Royale but missing in RapaGUI. The reason for this is of course that because of its cross-platform design RapaGUI represents the lowest common denominator between Windows, Linux, macOS, and AmigaOS. That's why not all MUI Royale features could be transferred to RapaGUI.

4 Tutorial

4.1 Tutorial

Welcome to the RapaGUI tutorial! This small step-by-step document will guide you through the process of creating your first GUI with RapaGUI in very few steps. We will create a little GUI application that consists of a listview, a text entry widget, and two buttons. The two buttons should be programmed to allow the addition and removal of items from the listview. The data to be inserted into the listview should be taken from the text entry widget. Here is what this little application looks like on Windows:



Let us start with the basics: In RapaGUI 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 RapaGUI GUI that contains a window with a listview, a text entry widget, and two buttons in XML format:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application>
   <window title="Example GUI">
      <vgroup>
         <listview>
            <column/>
         </listview>
         <textentry/>
         <hgroup>
            <button id="add">Add</button>
            <button id="rem">Remove</button>
         </hgroup>
      </vgroup>
   </window>
</application>
```

The `<application>` object is the master MOAI object for every application and must only be used once per application. All other MOAI objects are children of Application class. The root element of every `<window>` object must be derived from Group class, i.e. it must

be either `<vgroup>`, `<hgroup>`, or `<colgroup>`. Also note that there must be only one root element per window.

To see how our XML declaration above looks as a GUI, we have to save it to a file named `GUI.xml` and then use the following code to make RapaGUI convert it to a native GUI for the operating system Hollywood is running on:

```
@REQUIRE "RapaGUI"

moai.CreateApp(FileToString("GUI.xml"))

Repeat
   WaitEvent
Forever
```

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

```
@APPTITLE "Tutorial"
@APPVERSION "$VER: Tutorial 1.0 (29.12.15)"
@APPCOPYRIGHT "Copyright ©2015, Andreas Falkenhahn"
@APPAUTHOR "Andreas Falkenhahn"
@APPDESCRIPTION "The tutorial app from the RapaGUI guide"
```

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 RapaGUI event comes in. Thus, we have to modify our code as follows:

```
@REQUIRE "RapaGUI"

Function p_EventFunc(msg)
   ; contents follow below
EndFunction

InstallEventHandler({RapaGUI = p_EventFunc})
moai.CreateApp(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 RapaGUI event comes in. When we get such an event, we then have to check which MOAI 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.

The next thing we want to do is add the functionality that whenever the user presses the "Add" button the text in the text entry widget should get added to the listview as the last

entry. To do this, we first have to find a way of identifying our widgets 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 MOAI objects from Hollywood scripts. So let's add some IDs now for all widgets that we need to talk to. We have to modify our XML declaration like this:

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

Now that we have done this we can add some code to our event handler callback that grabs the contents of the text entry widget and adds it to the end of the list in our listview object. This is done by first calling `moai.Get()` on the `Textentry.Text` attribute to get the contents of the string widget and then running the method `Listview.Insert` on the listview widget using `moai.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$ = moai.Get("mystring", "text")
         moai.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
            moai.DoMethod("mylistview", "remove", "active")
         EndSwitch
```

Now we want the active entry of the listview to be automatically displayed in the text entry widget. For this purpose we have to set up 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 text entry widgets contents using the `Textentry.Text` attribute. Here is the code for doing that:

```
Switch msg.Class
...
Case "Listview":
   Switch msg.Attribute
   Case "Active":
      Local s$ = moai.DoMethod("mylistview", "getentry", "active")
      moai.Set("mystring", "text", 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 widgets 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:

```
...
moai.CreateApp(FileToString("GUI.xml"))
moai.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" button. 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$ = moai.DoMethod("mylistview", "getentry", "active")
      moai.Set("mystring", "text", s$)
      moai.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 `moai.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 RapaGUI programs should have a menu item informs the user that this program was done using RapaGUI. You can popup the this dialog by running the `Application.AboutRapaGUI` method on the application object. To add this menu to our program, we use Menubar class. Here is the XML code that you have to add before your window declaration:

```
...
<application>
   <menubar id="mymenubar">
      <menu title="File">
         <item id="menabout">About...</item>
         <item id="menaboutrapagui">About RapaGUI...</item>
         <item/>
         <item id="menquit">Quit</item>
      </menu>
   </menubar>
   ...
</application>
```

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

```
<window title="Example GUI" menubar="mymenubar">
```

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 `moai.DoMethod()` on it. Here is how the XML code needs to be adapted:

```
<application 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":
            moai.Request("Test", "Test program\n" ..
                "© 2015 by Andreas Falkenhahn", "OK")
         Case "menaboutrapagui":
            moai.DoMethod("app", "aboutrapagui")
         Case "menquit":
            End
```

```
            EndSwitch
         EndSwitch
      EndSwitch
```

Some final touches to our program could be adding online help to our widgets by using the `Area.Tooltip` and `Menuitem.Help` attributes, adding a status bar, and adding keyboard shortcuts for the menu items using the underscore character. These changes are left as an exercise for the reader.

So here is what our final program looks like. First the XML file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<application id="app">
   <menubar id="mymenubar">
      <menu title="File">
         <item id="menabout">About...</item>
         <item id="menaboutrapagui">
            About RapaGUI...</item>
         <item/>
         <item id="menquit">Quit</item>
      </menu>
   </menubar>
   <window title="Example GUI" menubar="mymenubar">
      <vgroup>
        <listview id="mylistview" notify="active">
            <column/>
         </listview>
         <textentry id="mystring"/>
         <hgroup>
            <button id="mybt1">Add</button>
            <button id="mybt2">Remove</button>
         </hgroup>
      </vgroup>
   </window>
</application>
```

And here is the code for the program logic:

```
@REQUIRE "RapaGUI"

@APPTITLE "Tutorial"
@APPVERSION "$VER: Tutorial 1.0 (29.12.15)"
@APPCOPYRIGHT "Copyright ©2015, Andreas Falkenhahn"
@APPAUTHOR "Andreas Falkenhahn"
@APPDESCRIPTION "The tutorial app from the RapaGUI guide"

Function p_EventFunc(msg)

   Switch msg.Class
   Case "Button":
      Switch msg.Attribute
```

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

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

   Case "Menuitem":
      Switch msg.Attribute
      Case "Selected":
         Switch msg.id
         Case "menabout":
            moai.Request("Test", "Test program\n" ..
                "© 2015 by Andreas Falkenhahn", "OK")
         Case "menaboutrapagui":
            moai.DoMethod("app", "aboutrapagui")
         Case "menquit":
            End
         EndSwitch
      EndSwitch

EndFunction

InstallEventHandler({RapaGUI = p_EventFunc})
moai.CreateApp(FileToString("GUI.xml"))
moai.Set("mybt2", "disabled", True)

Repeat
   WaitEvent
Forever
```

That's it! Now you should be able to create fantastic new GUI programs with Hollywood and RapaGUI. Thank you for reading this tutorial and enjoy the power of modern GUI programming with Hollywood and RapaGUI at your hands!

5 Examples

5.1 Examples

RapaGUI comes with a number of examples that demonstrate certain features and should allow you to get started really quickly. Here's a list of examples that are distributed with RapaGUI:

Accelerators
: Shows how to set up an accelerator table for advanced keyboard control of your GUI. See Section 7.1 [Accelerator class], page 57, for details.

Demo
: An extensive showcase of most of the widgets supported by RapaGUI. Take a look at this example to see how the different MOAI classes offered by RapaGUI appear as widgets.

Dialogs
: Shows how to use modal dialogs with RapaGUI. This example demonstrates both function-driven dialogs as well as normal modal dialogs. See Section 16.1 [Dialog class], page 95, for details.

DragNDrop
: Shows how to implement drag'n'drop using RapaGUI.

Dynamic1
: Demonstrates how to create new windows at runtime.

Dynamic2
: Demonstrates how to create new buttons at runtime.

Dynamic3
: Demonstrates how to create new tabs for Pageview class at runtime. See Section 34.1 [Pageview class], page 189, for details.

Dynamic4
: Demonstrates how to create menus and menu items at runtime.

HTMLView
: This is a rudimentary Internet browser written in RapaGUI and Hollywood. It demonstrates HTMLview class. See Section 23.1 [HTMLview class], page 129, for details.

Image
: This example shows how to embed a Hollywood brush inside your GUI using Image class. See Section 25.1 [Image class], page 135, for details.

MultiDisplays
: Demonstrates how to embed four different Hollywood displays inside a single RapaGUI window. This shows the magic that is possible when combining Hollywood and RapaGUI. See Section 20.1 [Hollywood class], page 121, for details.

Pages
: Shows how to create a window with tabbed groups using a Pageview class MOAI object. See Section 34.1 [Pageview class], page 189, for details.

Scrollbar
: Demonstrates how to attach a scrollbar created using Scrollbar class to a custom-drawn widget. See Section 42.1 [Scrollbar class], page 211, for details. Alternatively, you can also use Scrollcanvas class to achieve the same thing. See below for an example.

Scrollcanvas
: This example demonstrates another way of using scrollbars and widgets with custom graphics in RapaGUI. This time Scrollcanvas class is used to See Section 43.1 [Scrollcanvas class], page 215, for details. The same thing is also possible by using Scrollbar class. See above for an example.

ShowHide
: Shows how to show and hide MOAI objects at runtime by setting the `Area.Hide` attribute.

SongPlayer
: This is an audio file player written in RapaGUI.

TextEditor
: A little text editor written in Hollywood. It also demonstrates how to use a toolbar and a status bar.

Treeview
: Demonstrates how to create a multi-column treeview in RapaGUI using Treeview class. See Section 54.1 [Treeview class], page 263, for details.

VideoPlayer
: This is a video player written in RapaGUI. It shows how to embed a Hollywood display inside a RapaGUI window using Hollywood class. See Section 20.1 [Hollywood class], page 121, for details. Furthermore, it also demonstrates how to use image buttons, a listview with images and slider objects.

6 Function reference

6.1 moai.CreateApp

NAME

moai.CreateApp – create application object from an XML source

SYNOPSIS

`moai.CreateApp(xml$)`

FUNCTION

This function creates an application from the XML description passed in the `xml$` argument. Please note that `xml$` 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 MOAI objects that you have defined in the XML GUI declaration using the `moai.Set()`, `moai.Get()` and `moai.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 `moai.CreateApp()` a second time, you have to free the old GUI first using the `moai.FreeApp()` call.

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

INPUTS

`xml$` a string containing an XML GUI description

EXAMPLE

```
moai.CreateApp([[
<?xml version="1.0" encoding="iso-8859-1"?>
<application>
   <window title="Test program">
      <vgroup>
         <button id="btn">Hello World!</button>
      </vgroup>
   </window>
</application>
]])

InstallEventHandler({RapaGUI = Function(msg)
      If msg.attribute = "Pressed" Then DebugPrint("Button pressed!")
   EndFunction})

Repeat
   WaitEvent
```

```
Forever
```

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

6.2 moai.CreateDialog

NAME

moai.CreateDialog – create dialog object from an XML source

SYNOPSIS

```
moai.CreateDialog(xml$[, parent$])
```

FUNCTION

This function can be used to dynamically create a dialog object from an XML source. The newly created dialog object will also be automatically attached to your application so that it is ready for instant use. You should also specify a parent window for your dialog using the optional argument.

`moai.CreateDialog()` is very important because you should create dialogs only when you need them and destroy them as soon as you are finished with them. It is not advised to create all your dialogs on startup using `moai.CreateApp()` and keep them in memory all the time. Instead, you should use this function to create a dialog when you need it and then have it destroyed as soon as you are finished with it. The reason for this is that windows are quite a finite resource on some operating systems supported by RapaGUI. For example, on Windows there is a limit of about 10,000 windows per process. That might sound like sufficiently enough but keep in mind that on Windows every widget is a "window" for the operating system, e.g. every label, button, frame, checkbox, group, etc. in your application is a window so you should take care that you create your dialogs only as needed using `moai.CreateDialog()` and destroy them right afterwards.

In practice, it is advised to create a separate XML file for each of your dialogs and then use this function to convert the XML file into a dialog at runtime and have the dialog destroyed automatically by RapaGUI as soon as the user closes it.

Technically speaking, this function is just a convenience function which internally calls `moai.CreateObject()` and then adds the newly created object to the application object by calling `Application.AddWindow`. `moai.CreateDialog()` simply combines these two steps into one.

See Section 16.1 [Dialog class], page 95, for more information on dialogs.

INPUTS

`xml$`	a string containing an XML MOAI dialog description
`parent$`	optional: desired parent for the dialog object; see above for details

EXAMPLE

```
moai.CreateDialog([[
<dialog id="dlg" title="Question">
  <vgroup>
      <text>What is your name?</text>
      <textentry/>
```

```
      <hgroup>
         <button id="ok">OK</button>
         <button id="cancel">Cancel</button>
      </hgroup>
   </vgroup>
</dialog>
]])

moai.DoMethod("dlg", "showmodal")
```

The code above creates a new dialog and shows it.

6.3 moai.CreateObject

NAME

moai.CreateObject – create MOAI object from an XML source

SYNOPSIS

`moai.CreateObject(xml$[, parent$])`

FUNCTION

This function can be used to dynamically create a MOAI object from an XML source. When `moai.CreateObject()` returns, the newly created MOAI object won't be attached to any parent object and will live in a state of isolation from your application object created by `moai.CreateApp()`. 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 MOAI 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 `Menubar.Prepend`, `Menubar.Append`, `Menubar.Insert`, `Menu.Prepend`, `Menu.Append`, or `Menu.Insert` methods. All other objects can be attached by using the group methods `Group.Prepend`, `Group.Append` and `Group.Insert`.

For most MOAI classes you have to specify a parent object in the second parameter. RapaGUI won't attach the new object to the parent but it still needs to know the parent for determining certain settings. The only MOAI classes which do not require you to specify a parent object are the following:

- Window class
- Dialog class
- Menubar class
- Menu class
- Menuitem class

All other MOAI classes require you to specify the identifier of the parent object.

Normally, you just have to pass the identifier of the window that you want to attach the MOAI object to as the parent. There is one exception: If you plan to add the MOAI object to a group with a frame, i.e. an instance of Group class with `Group.Frame` set to `True`, then you have to pass this group object as the parent. Note that this

only applies to groups with a frame. If you plan to add the MOAI object to a normal group that doesn't have a frame, you just have to pass the identifier of the window to `moai.CreateObject()`. Note, though, that in case the normal group is itself just a child embedded somewhere in the hierarchy of a framed group, then you have to pass the identifier of the framed group. Also, when adding objects to scrollgroups, you need to pass the scrollgroup as the parent.

To put it in abstract terms: The parent must be set to the next object in the layout hierarchy that has a visual representation. Normal groups are just layout tools, they don't exist as widgets. Framed groups, however, are layout tools but they are also widgets since they have a visual representation. Thus, if your object is going to be embedded somewhere in a framed group, the framed group must be passed as the parent. If it is embedded in a normal group, then the top-level window must be the parent. Don't forget to think in hierarchies: Even if you want to attach your object to a normal group it could still be the case that the normal group is itself embedded in a framed group which means that you would still have to pass the framed group as the parent. The same applies to scrollgroups since they also have a visual representation.

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

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

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

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

INPUTS

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

`parent$` optional: desired parent for the object; see above for details

EXAMPLE

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

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

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

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

moai.DoMethod("mygroup", "initchange")
moai.DoMethod("mygroup", "append", "newbutton")
moai.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".

6.4 moai.DoMethod

NAME

moai.DoMethod – run method on MOAI object

SYNOPSIS

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

FUNCTION

This function can be used to run a method on the specified MOAI object. You have to pass the identifier of the MOAI 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 MOAI object. Have a look at the class reference to see what methods are supported by the different MOAI 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$`	identifier of MOAI object to run method on
`method$`	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

```
moai.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.

6.5 moai.FreeApp

NAME
moai.FreeApp – delete entire application object

SYNOPSIS
`moai.FreeApp()`

FUNCTION
Use this call to delete the entire application object created using the last call to `moai.CreateApp()`. `moai.FreeApp()` will delete the whole application object and all of the children attached to it. After this call returns, you could create a new application using the `moai.CreateApp()` function if you want.

Note that `moai.FreeApp()` won't free MOAI objects which are currently in a detached state. Those have to be freed by using `moai.FreeObject()`.

INPUTS
none

6.6 moai.FreeDialog

NAME
moai.FreeDialog – delete a dialog object

SYNOPSIS
`moai.FreeDialog(id$)`

FUNCTION
This function can be used to delete a dialog object created by `moai.CreateDialog()`. Normally, it isn't necessary to call this function because dialogs are automatically destroyed by `Dialog.EndModal` or when the dialog function returns. If you need fine-tuned control over dialog destruction, you can use this function, though.

`moai.FreeDialog()` will first detach the dialog from the application object and then free it. So technically speaking, this function is just a convenience function which internally removes the dialog object from the application object by calling `Application.RemoveWindow` and then calls `moai.FreeObject()` to free the dialog object. `moai.FreeDialog()` simply combines these two steps into one.

See Section 16.1 [Dialog class], page 95, for more information on dialogs.

INPUTS

`id$`	identifier of dialog object to free

6.7 moai.FreeImage

NAME
moai.FreeImage – free image in cache (V1.2)

SYNOPSIS
`moai.FreeImage(id[, isicon])`

FUNCTION

This function can be used to free an image in RapaGUI's internal cache. You have to pass the identifier of the image that should be freed in `id`. If the optional `isicon` argument is set to `True`, `id` must be the identifier of a Hollywood icon to free. Otherwise, `id` must be the identifier of a Hollywood brush that should be freed. Alternatively, you can also pass -1 in `id` to free all images in RapaGUI's internal cache.

You must make sure that the specified image 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 cached images are freed automatically by RapaGUI. Under certain conditions, however, it can be useful to call this function.

RapaGUI caches all images that you use in your GUI. That is why when you try to use a certain image in your GUI a second time, it will just be loaded from RapaGUI's internal image cache for performance reasons. This can lead to unwanted behaviour in case you have updated your image's graphics in the meantime and you want RapaGUI to use the updated graphics. In that case, you first have to free the image in RapaGUI's internal image cache by using this function. When you pass the image to RapaGUI again then, it will be re-created from the image's current contents and it will be cached anew. For listview and treeview widgets, you can also use `moai.UpdateImage()` to update the graphics data of an image.

See Section 3.20 [Image cache], page 30, for details.

INPUTS

`id`	identifier of image to free or -1 to free all images
`isicon`	optional: `True` if `id` contains the identifier of an icon, `False` if it contains the identifier of a brush (defaults to `False`) (V2.0)

6.8 moai.FreeObject

NAME

moai.FreeObject – delete a detached MOAI object

SYNOPSIS

```
moai.FreeObject(id$)
```

FUNCTION

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

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

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

INPUTS

`id$` identifier of MOAI object to free

6.9 moai.Get

NAME

moai.Get – get value of a MOAI object attribute

SYNOPSIS

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

FUNCTION

This function can be used to retrieve the current value of the attribute `attr$` in the MOAI object specified in `id$`. 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 MOAI object. Have a look at the class reference to see what attributes are supported by the different MOAI classes. In order to use an attribute with this function, it needs to have an applicability of `"G"`. Attributes of Area class and MOAI class can be used on almost all other classes because the Area and MOAI classes act as superclasses for most of the other classes.

INPUTS

`id$` identifier of MOAI object to query

`attr$` attribute whose value should be retrieved

RESULTS

`r` current attribute value

EXAMPLE

```
DebugPrint(moai.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.

6.10 moai.HaveObject

NAME

moai.HaveObject – check if MOAI object exists

SYNOPSIS

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

FUNCTION

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

INPUTS

`id$` MOAI object id to check

RESULTS

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

6.11 moai.Notify

NAME

moai.Notify – add/remove notification on a MOAI object attribute

SYNOPSIS

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

FUNCTION

This function can be used to add or remove a notification on the attribute `attr$` in the MOAI object specified in `id$`. 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 MOAI object. Have a look at the class reference to see what attributes are supported by the different MOAI classes. In order to use an attribute with this function, it needs to have an applicability of `"N"`. Attributes of Area class and MOAI class can be used on almost all other classes because the Area and MOAI 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 RapaGUI event handler callback using the `InstallEventHandler()` Hollywood function. See Section 3.7 [Notifications], page 13, for details.

Please note that notifications can also be setup in the XML GUI declaration by using the `Notify` attribute. See Section 3.7 [Notifications], page 13, for details.

INPUTS

`id$` identifier of MOAI object to use

`attr$` attribute to listen to

`enable` `True` to add a notification or `False` to remove notification from this object

EXAMPLE

```
moai.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"`.

6.12 moai.Request

NAME

moai.Request – show a system requester

SYNOPSIS

```
r = moai.Request(title$, body$, buts$[, icon$])
```

FUNCTION

This function pops up a standard system requester that displays a message (`body$`) and also allows the user to make a selection using one of the buttons specified by `buts$`. Optionally, you can also specify an icon that should be displayed in the requester. If you specify an empty string in `title$`, it will automatically use the title specified in `@APPTITLE`.

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

The optional argument `icon$` can be set to one of the following predefined values:

`"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
`buts$`	one or more buttons that the user can press
`icon$`	optional: icon to show in the requester (defaults to "Information")

RESULTS

`r`	the button that was pressed by the user

EXAMPLE

```
moai.Request("RapaGUI", "Hello!\n\n" ..
    "Do you like RapaGUI!", "Yes|No")
```

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

6.13 moai.Set

NAME

moai.Set – set value of a MOAI object attribute

SYNOPSIS

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

FUNCTION

This function can be used to set the current value of one or more attributes in the MOAI 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 `moai.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 MOAI object. Have a look at the class reference to see what attributes are supported by the different MOAI classes. In order to use an attribute with this function, it needs to have an applicability of "S". Attributes of Area class and MOAI class can be used on almost all other classes because the Area and MOAI 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 `moai.Set()` on that attribute. If you do not want this behaviour, you can use the `MOAI.NoNotify` attribute to prevent a notification from being issued.

INPUTS

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

EXAMPLE

```
moai.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.

```
moai.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 `MOAI.NoNotify` attribute to `True`. This is useful if you need to distinguish between user selections in the listview and selections made programmatically using `moai.Set()`.

6.14 moai.UpdateImage

NAME

moai.UpdateImage – update cached image (V2.0)

SYNOPSIS

```
moai.UpdateImage(id[, isicon])
```

FUNCTION

This function can be used to update the graphics data of an image in RapaGUI's internal cache. You have to pass the identifier of the image that should be updated in `id`. The

optional parameter `isicon` specifies if the cached image is a Hollywood brush or icon. The new graphics data for the image will also be taken from the Hollywood brush or icon specified by `id`.

Note that this function is currently only compatible with listview and treeview widgets. You cannot use it to update the graphics of icons in other widgets, e.g. icons in a button widget. After updating the graphics data of an icon in a listview or treeview widget, you also need to make sure that all rows that use that icon are refreshed. You can do that by using the `Listview.Rename` or `Treeviewleaf.SetItem` methods, for example.

See Section 3.20 [Image cache], page 30, for details.

INPUTS

`id`	identifier of image to be updated
`isicon`	optional: `True` if `id` contains the identifier of an icon, `False` if it contains the identifier of a brush (defaults to `False`)

7 Accelerator class

7.1 Overview

Accelerator class can be used to set up global keyboard shortcuts for a window. Normal keyboard shortcuts defined by using the underscore character are handled automatically by RapaGUI. See Section 3.13 [Keyboard shortcuts], page 21, for details. However, sometimes you might want to use more complex keyboard shortcuts, e.g. you might want to listen to the escape key, function keys, certain control keys or key combinations (e.g. `CTRL+V` for paste). This is possible by using Accelerator class.

Accelerator class allows you to set up a table of keyboard shortcuts which can then be assigned to a window by using the `Window.Accelerator` attribute to make those shortcuts globally available in the respective window. Each accelerator table must contain at least one child of type Acceleratoritem class defining a single shortcut.

Here is an example of how to set up an accelerator table in XML:

```
<accelerator>
   <item id="ac_cut" mod="ctrl">X</item>
   <item id="ac_copy" mod="ctrl">C</item>
   <item id="ac_paste" mod="ctrl">V</item>
   <item id="ac_f1">F1</item>
</accelerator>
```

The accelerator table declared above sets up four different shortcuts: `CTRL+X`, `CTRL+C`, `CTRL+V` and `F1`. Whenever the user presses such a key or key combination, the `Acceleratoritem.Pressed` notification will be triggered and your application can then react on it.

Note that if you intend to link your accelerator items to menu items, you don't have to use Accelerator class at all. In that case you can just use the `Menuitem.Shortcut` attribute to set up shortcuts for your menu items. Accelerator class is really only necessary if you need to listen to keyboard events without having any menu items involved.

Accelerator class doesn't define any attributes or methods itself. See Section 8.1 [Acceleratoritem class], page 59, for all necessary information.

8 Acceleratoritem class

8.1 Overview

Acceleratoritem class is used to create an individual keyboard shortcut as part of an accelerator table for Accelerator class. You cannot create independent instances of Acceleratoritem class. They always need to be embedded inside Accelerator class. See Section 7.1 [Accelerator class], page 57, for details.

The following keys can currently be specified when creating an accelerator item using the `<item>` XML tag:

`UP`	Cursor up
`DOWN`	Cursor down
`RIGHT`	Cursor right
`LEFT`	Cursor left
`HELP`	Help key
`DEL`	Delete key
`BACKSPACE`	Backspace key
`TAB`	Tab key
`RETURN`	Return key
`ENTER`	Enter key
`ESC`	Escape
`SPACE`	Space key
`F1 - F16`	Function keys
`INSERT`	Insert key
`HOME`	Home key
`END`	End key
`PAGEUP`	Page up key
`PAGEDOWN`	Page down key
`PRINT`	Print key
`PAUSE`	Pause key

Furthermore, Acceleratoritem class supports the English alphabet characters from A to Z as well as the numbers 0 to 9 as key specifications. It doesn't matter whether you specify all these keys in upper, lower or mixed case. See Section 7.1 [Accelerator class], page 57, for an example.

8.2 Acceleratoritem.Mod

NAME

Acceleratoritem.Mod – set modifier key(s)

FUNCTION

This attribute allows you to specify the modifier key(s) for this accelerator item. This can be a combination of the following predefined values:

`None`	Don't use any modifier key. This is the default.
`Ctrl`	Add the control key to the modifier keys.
`Alt`	Add the alt key to the modifier keys.
`Cmd`	Add the command key to the modifier keys. This is only supported on macOS and AmigaOS.
`Shift`	Add the shift key to the modifier keys.

If you specify more than one modifier key, you have to separate the individual keys by a semicolon, e.g. `"ctrl;shift"`.

When using modifier keys, the keyboard shortcut will only trigger if the modifier key(s) and the main key are both down. This is useful for monitoring key combinations like `CTRL+V` for pasting data, etc.

TYPE

String

APPLICABILITY

I

8.3 Acceleratoritem.Pressed

NAME

Acceleratoritem.Pressed – learn when the shortcut is pressed

FUNCTION

This attribute is triggered if the user presses the key(s) defined by this accelerator item. RapaGUI automatically listens to this attribute so you do not need to explicitly request a notification using the MOAI.Notify attribute.

TYPE

Boolean

APPLICABILITY

N

9 Application class

9.1 Overview

Application class is the master class that manages an application. Thus, there can be only one instance of this class in every application. This instance is created by a call to `moai.CreateApp()`. All windows that an application shows are children of the application class. Thus, every GUI definition has to start with the application class.

9.2 Application.AboutMUI

NAME

Application.AboutMUI – show the MUI about window

SYNOPSIS

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

PLATFORMS

AmigaOS and compatibles only

FUNCTION

Show the MUI about window. The MUI styleguide says that all MUI applications should include a menu item called "About MUI...".

INPUTS

`id` id of the application object

9.3 Application.AboutRapaGUI

NAME

Application.AboutRapaGUI – show the RapaGUI about window

SYNOPSIS

```
moai.DoMethod(id, "AboutRapaGUI")
```

FUNCTION

This method shows the RapaGUI about window. You should include a menu item called "About RapaGUI..." in all your applications.

INPUTS

`id` id of the application object

9.4 Application.AddWindow

NAME

Application.AddWindow – add detached window object to application object

SYNOPSIS

```
moai.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 `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Application.RemoveWindow` method.

Please do note that you must not call `Application.AddWindow` for dialogs created using `moai.CreateDialog()` since `moai.CreateDialog()` already calls `Application.AddWindow` internally.

INPUTS

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

EXAMPLE

See Section 6.3 [moai.CreateObject], page 47.

9.5 Application.ContextMenu

NAME

Application.ContextMenu – set context menu for application (V2.0)

PLATFORMS

Windows, macOS, Linux

FUNCTION

If your application has installed an icon in the system tray by using Hollywood's `SetTrayIcon()` function, you can use this attribute to add a context menu to the application's tray icon. Just set this attribute to an identifier of a MOAI object derived from Menu class and your application's icon in the system tray will get a context menu. Whenever the user presses the right mouse button on the icon in the system tray, the context menu will be shown.

See Section 3.16 [Context menus], page 24, for an example.

TYPE

MOAI object

APPLICABILITY

I

9.6 Application.HelpFile

NAME

Application.HelpFile – set/get application help file (V2.0)

FUNCTION

Set a help file for the application. On Windows this should point to a CHM file, on Amiga to an AmigaGuide file and on all other platforms to a directory which contains all HTML files belonging to the application's manual (the main file must be called `index.html`, the index file must be called `findex.html`) help file.

Once you have defined a help file, you can then use `Application.ShowHelp` and `Application.ShowHelpNode` to open the help file.

This is currently unsupported on Android.

TYPE

String

APPLICABILITY

ISG

9.7 Application.Icon

NAME

Application.Icon – set application icon

PLATFORMS

AmigaOS and compatibles only

FUNCTION

This tag allows you to set the icon that will be shown on Workbench screen whenever the application is iconified. You need to set this tag to the filename of an Amiga `.info` icon file.

Normally, you should set the icons for your application using Hollywood's `@APPICON` preprocessor command. This tag is only here to support Amiga appicons on Workbench.

TYPE

String

APPLICABILITY

I

9.8 Application.OpenConfigWindow

NAME

Application.OpenConfigWindow – show MUI preferences window

SYNOPSIS

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

PLATFORMS

AmigaOS and compatibles only

FUNCTION

Make MUI open the preferences window for the application. MUI supports individual user interface settings for each application which is why every MUI application should add a menu item like "Settings/MUI..." to allow the user to configure these settings.

INPUTS

`id`	id of the application object

9.9 Application.RemoveWindow

NAME

Application.RemoveWindow – detach window object from application object

SYNOPSIS

```
moai.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 `moai.FreeObject()`.

Please do note that you must not call `Application.RemoveWindow` for dialogs freed using `moai.FreeDialog()` since `moai.FreeDialog()` already calls `Application.RemoveWindow` internally.

INPUTS

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

9.10 Application.Sleep

NAME

Application.Sleep – put application to sleep

FUNCTION

This attribute will put the application to sleep. All open windows will be disabled and they won't accept user input any more. This can be useful in case your application is currently loading or saving files which makes it unable to handle user input.

TYPE

Boolean

APPLICABILITY

S

9.11 Application.ShowHelp

NAME

Application.ShowHelp – show application help file (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "ShowHelp", what$)
```

FUNCTION

If you have defined a help file for your application using the `Application.HelpFile` attribute, you can use this method to show that help file. You have to pass the part of the help file that should be shown in the `what$` parameter. This can be one of the following special values:

`Contents` Show the help file's table of contents.

`Index` Show the help file's index.

`Search` Show the help file's search page. This is currently only supported on Windows.

To show a specific node of the help file, use the `Application.ShowHelpNode` method instead.

INPUTS

`id` id of the application object

`what$` part of the help file to show (see above for possible values)

9.12 Application.ShowHelpNode

NAME

Application.ShowHelpNode – show certain node in help file (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "ShowHelpNode", node$)
```

FUNCTION

If you have defined a help file for your application using the `Application.HelpFile` attribute, you can use this method to show the node specified by `node$` in that help file.

To show one of the standard parts of the help file instead of a specific node, use the `Application.ShowHelp` method instead.

INPUTS

`id` id of the application object

`node$` node to show

9.13 Application.UseIcons

NAME

Application.UseIcons – set default image type to icon (V2.0)

FUNCTION

This attribute can be used to globally change the default image type for attributes such as `Button.Icon` or `Toolbarbutton.Icon` to icon instead of brush. Even though the names of those attributes might suggest otherwise, they indeed expect a Hollywood brush by default for historical reasons. Using Hollywood icons, however, is often more

convenient because they offer greater flexibility, especially when it comes to high-DPI support, which is why using Hollywood icons instead of brushes is often preferred by GUI designers nowadays. See Section 3.12 [High-DPI support], page 20, for details.

Alternatively, you can also change the default image type from brush to icon on a per-widget basis by using attributes such as `Button.IconType` or `Toolbarbutton.IconType` but doing this globally using `Application.UseIcons` might be more convenient.

TYPE
Boolean

APPLICABILITY
I

9.14 Application.WindowMenu

NAME
Application.WindowMenu – set window menu title (V2.0)

PLATFORMS
macOS

FUNCTION
Set the title of macOS' standard window menu to a custom one. This is useful when localizing your applications.

TYPE
String

APPLICABILITY
I

10 Area class

10.1 Overview

Area class is the super class of every MOAI class that creates objects which have a visual representation within a window. As such it allows you to manage several generic attributes like object visibility (shown or hidden), state (enabled or disabled), tooltips, and context menus. Children which are derived from Area class are usually called widgets (a portmanteau of "window" and "gadget"). Area class can also provide you with information about a widget's position and size as well as other details belonging to its visual representation within a window.

Note that objects deriving from Group class aren't children of Area class since Group class just creates groups of widgets but isn't a widget itself, but just a layout tool.

10.2 Area.ContextMenu

NAME

Area.ContextMenu – set context menu for object

FUNCTION

Set this attribute to an identifier of a MOAI object derived from Menu class and the respective object will get a context menu. Whenever the user presses the right mouse button on the respective object, the context menu will be shown.

Also note that context menu events will then be delivered via the standard `Menuitem.Selected` mechanism and not through a special context menu event handler. Since you can use the same menu object on several widgets, you need a way to find out the widget whose context menu triggered the event. To give you this information, the event message will contain an additional item named `Parent` which contains the ID of the context menu's hosting widget. This allows you to re-use the same menu object as a context menu for multiple parent widgets.

Starting with RapaGUI 2.0 you can also set up a notification on the attribute. In that case, your event handler will be called immediately before RapaGUI shows the context menu. You can then return either `False` for no context menu or use `moai.Set()` on `Area.ContextMenu` to set the context menu that should be shown. This is very useful for customizing the context menu depending on where the right mouse button has been clicked. The position will be passed to your event handler. In case you are listening to `Area.ContextMenu`, your event handler will also be passed some additional information depending on the context menu's parent widget. Here is an overview of those additional message fields:

`X` X coordinate of mouse click.

`Y` Y coordinate of mouse click.

`CursorPos`
Cursor index at the position where the user clicked. This is only set for widgets of type Texteditor class.

`Row` Row at the position of the mouse click or -1 if there is no row at the click position. This is only set for widgets of type Listview class.

`Column` Column at the position of the mouse click or -1 if there is no column at the click position. This is only set for widgets of type Listview class.

`Item` Treeview item at the position of the mouse click or -1 if there is no item at the click position. This is only set for widgets of type Treeview class.

AmigaOS users please note that MUI doesn't allow context menu objects to be shared across windows. You are not allowed to use the same menu object with widgets in different windows. It's okay to re-use a menu object for several widgets in the same window but not for widgets in another window. All other platforms don't have this limitation, only AmigaOS is affected here.

To remove a context menu completely, pass the special string `"(none)"` to this attribute.

See Section 3.16 [Context menus], page 24, for an example.

TYPE
MOAI object

APPLICABILITY
ISGN

10.3 Area.Disabled

NAME
Area.Disabled – set/get disabled state

FUNCTION
Set this attribute to disable or enable a widget. Disabled widgets do not accept user input any longer and are visually recognizable as disabled objects.

TYPE
Boolean

APPLICABILITY
ISG

10.4 Area.FixHeight

NAME
Area.FixHeight – fix height of object

FUNCTION
Set this to `True` to disable vertical resizing for this object.

TYPE
Boolean

APPLICABILITY
I

10.5 Area.FixWidth

NAME
Area.FixWidth – fix width of object

FUNCTION
Set this to `True` to disable horizontal resizing for this object.

TYPE
Boolean

APPLICABILITY
I

10.6 Area.FontName

NAME
Area.FontName – set/get font for object (V2.0)

FUNCTION
Set or get the font for the object.

Note that on AmigaOS and compatibles this can only be set at initialization time. You cannot change fonts for existing objects. Also, on AmigaOS you must also set `Area.FontSize` if you set this attribute.

Also note that on AmigaOS `Area.FontName` can't be used with widgets of type Label class. Use Text class class.

TYPE
String

APPLICABILITY
ISG

10.7 Area.FontSize

NAME
Area.FontSize – set/get font size for object (V2.0)

FUNCTION
Set or get the font size for the object. The font size must be either a value in points (non-Amiga) or in pixels (Amiga) or one of the following predefined font sizes:

`Normal` Normal font size.

`Small` Small font size.

`Big` Big font size.

Note that on AmigaOS and compatibles this can only be set at initialization time. You cannot change fonts for existing objects. Also, on AmigaOS you must also set

`Area.FontName` if you set this attribute to a numeric size. If you pass one of the predefined font sizes (see above) on AmigaOS, you must not set `Area.FontName`, though.

Also note that on AmigaOS `Area.FontSize` can't be used with widgets of type Label class. Use Text class class.

TYPE
Number or string (see above for possible values)

APPLICABILITY
ISG

10.8 Area.FontStyle

NAME
Area.FontStyle – set/get font style for object (V2.0)

FUNCTION
Set or get the font style for the object. This can be a combination of the following predefined styles:

`Normal` Normal font style. This cannot be combined with any other styles.

`Bold` Bold font style.

`Italic` Italic font style.

`Underlined`
Underlined font style.

`Fixed` Use a fixed width font.

Multiple styles can be combined by using a semicolon as a separator.

TYPE
String (see above for possible values)

APPLICABILITY
ISG

10.9 Area.Height

NAME
Area.Height – set/get height of object

FUNCTION
Set this attribute to the desired object height in device-independent pixels. This is normally not necessary because RapaGUI automatically chooses an appropriate size for its objects. However, in some cases it might be handy to have fine-tuned control over object sizes.

When the parent window of the object is open, you can also read the object's height using attribute.

TYPE
Number

APPLICABILITY
IG

10.10 Area.Hide

NAME
Area.Hide – show/hide object

FUNCTION
Set this attribute to show or hide an object.

TYPE
Boolean

APPLICABILITY
ISG

10.11 Area.Left

NAME
Area.Left – get left edge of object

FUNCTION
This attribute returns the x-position of an object within its parent window. This is only valid if the object is currently visible.

TYPE
Number

APPLICABILITY
G

10.12 Area.NoAutoKey

NAME
Area.NoAutoKey – disable automatic shortcut generation

FUNCTION
Set this attribute to disable automatic shortcut generation. By default, the character following an underscore character in a label or a menu entry will be treated as an accelerator key that can be used to access the widget using the keyboard. If you set this attribute, underscore characters will never be treated as shortcut markers and will be shown normally in labels and menu entries.

See Section 3.13 [Keyboard shortcuts], page 21, for details.

TYPE
Boolean

APPLICABILITY
I

10.13 Area.Redraw

NAME
Area.Redraw – force complete widget redraw

SYNOPSIS
```
moai.DoMethod(id, "Redraw")
```

FUNCTION
This method redraws the complete widget. Normally, this is not necessary since widgets decide for themselves when they need to be redrawn. Still, it might be useful for testing purposes or when using a widget derived from Scrollcanvas class.

INPUTS
`id` id of widget

10.14 Area.Tooltip

NAME
Area.Tooltip – set/get tooltip string

FUNCTION
Set a string that should be shown in a tooltip when the mouse hovers over the object.

TYPE
String

APPLICABILITY
ISG

10.15 Area.Top

NAME
Area.Top – get top edge of object

FUNCTION
This attribute returns the y-position of an object within its parent window. This is only valid if the object is currently visible.

TYPE
Number

APPLICABILITY
G

10.16 Area.Weight

NAME

Area.Weight – set object weight

FUNCTION

This attribute controls the weight of an object inside a group of objects. By default, all objects in a group have a weight of 100. If you want to have an object that appears twice as large, you have to give it a weight of 200. To make an object appear one and a half times as large, just use 150 and so on.

Naturally, this attribute is only useful for objects which are resizable.

Note that when setting `Area.Weight` you normally shouldn't set specific dimensions via `Area.Width` and `Area.Height`. The width and height should be determined by the `Area.Weight` setting instead.

TYPE

Number

APPLICABILITY

I

10.17 Area.Width

NAME

Area.Width – set/get width of object

FUNCTION

Set this attribute to the desired object width in device-independent pixels. This is normally not necessary because RapaGUI automatically chooses an appropriate size for its objects. However, in some cases it might be handy to have fine-tuned control over object sizes.

When the parent window of the object is open, you can also read the object's width using attribute.

TYPE

Number

APPLICABILITY

IG

11 Busybar class

11.1 Overview

Busybar class derives from Area class and creates a widget which shows an animation indicating that the application is currently in a busy state. This animation, however, just shows activity but no progress. If you would like to visualize progress, use Progressbar class instead. See Section 39.1 [Progressbar class], page 203, for details.

Once you've created an object of Busybar class, you have to call `Busybar.Move` repeatedly to move the animation. This can be achieved by setting up an interval timer using Hollywood's `SetInterval()` function.

11.2 Busybar.Move

NAME

Busybar.Move – draw next animation frame

SYNOPSIS

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

FUNCTION

You have to call this method repeatedly in order to animate the widget. A good idea is to install a Hollywood interval timer using `SetInterval()` and then call this method from the timer about 10 times per second or so.

INPUTS

`id` id of the busybar object

11.3 Busybar.Reset

NAME

Busybar.Reset – stop animation

SYNOPSIS

```
moai.DoMethod(id, "Reset")
```

FUNCTION

This method resets the animation. Successive calls to `Busybar.Move` will start drawing from the first frame again.

INPUTS

`id` id of the busybar object

12 Button class

12.1 Overview

Button class derives from Area class and allows you to create buttons very easily. The XML tag's content is used as the button label. If the button label contains an underscore, RapaGUI will automatically set up the character following this underscore as a keyboard shortcut. If you don't want this behaviour, set `Area.NoAutoKey` to `True`. See Section 3.13 [Keyboard shortcuts], page 21, for details.

Example:

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

12.2 Button.Hint

NAME
Button.Hint – set button hint (V2.0)

PLATFORMS
Android

FUNCTION
When using buttons in dialogs on Android, you can specify a hint value that tells Android whether the button is the positive, negative, or neutral one. This hint value can be set using this attribute. You can set it to the following values:

`None`	No hint. This is the default setting.
`Positive`	Button is the positive button in a dialog (e.g. `"Yes"`).
`Negative`	Button is the negative button in a dialog (e.g. `"No"`).
`Neutral`	Button is the neutral button in a dialog (e.g. `"Cancel"`).

TYPE
String (see above for possible values)

APPLICABILITY
I

12.3 Button.Icon

NAME
Button.Icon – set button image

FUNCTION
Set this attribute to the identifier of a Hollywood brush or icon to add an image to your button. Whether this attribute expects a Hollywood brush or icon, depends on what you specify in the `Button.IconType` attribute. By default, `Button.Icon` expects a Hollywood brush. You can control the position of the icon in relation to the button's label by setting the `Button.IconPos` attribute.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

Note that on AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

TYPE
Number

APPLICABILITY
I

12.4 Button.IconPos

NAME
Button.IconPos – define button icon position

FUNCTION
Set the desired position of an icon specified using `Button.Icon`. This can be one of the following values:

`Left` Show icon to the left of the label. This is the default.

`Right` Show icon to the right of the label.

Note that on AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

TYPE
String (see above for possible values)

APPLICABILITY
I

12.5 Button.IconScale

NAME
Button.IconScale – configure automatic image scaling (V2.0)

FUNCTION
If `Button.Icon` has been set to a raster brush and RapaGUI is running on a high-DPI display, RapaGUI will automatically scale the brush's raster graphics to fit to the current monitor's DPI setting. If you don't want that, set this tag to `False`.

Alternatively, you can also globally disable automatic image scaling by setting the `ScaleGUI` tag to `False` when calling `@REQUIRE` on RapaGUI. See Section 3.4 [Initializing RapaGUI], page 11, for details.

Please also read the chapter about high-DPI support in RapaGUI to learn more about supporting high-DPI displays. See Section 3.12 [High-DPI support], page 20, for details.

TYPE
Boolean

APPLICABILITY
I

12.6 Button.IconType

NAME
Button.IconType – set icon type to use (V2.0)

FUNCTION
This attribute allows you to set the type of the Hollywood image object passed in the `Button.Icon` attribute. By default, `Button.Icon` expects a Hollywood brush. By setting `Button.IconType`, however, you can make it use a different Hollywood image type.

The following image types are currently available:

`Brush` Use a Hollywood brush. This is the default type. You can use either raster or vector brushes. Vector brushes have the advantage that they can be scaled to any resolution without losses in quality. This is very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

`Icon` Use a Hollywood icon. This image type has the advantage that it can contain several subimages of different sizes. This makes it possible to provide images in different resolutions which can be very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

Note that you can globally change the default of all `IconType` attributes to Hollywood icons by setting the `Application.UseIcons` tag. See Section 9.13 [Application.UseIcons], page 65, for details.

TYPE
String (see above for possible values)

APPLICABILITY
I

12.7 Button.Pressed

NAME
Button.Pressed – learn if a button is pressed

FUNCTION
This attribute is triggered if the user presses the button. RapaGUI automatically listens to this attribute for all buttons so you do not need to explicitly request a notification using the MOAI.Notify attribute.

TYPE
Boolean

APPLICABILITY
N

12.8 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.

RapaGUI automatically listens to this attribute for all buttons so you do not need to explicitly request a notification using the MOAI.Notify attribute.

TYPE
Boolean

APPLICABILITY
ISGN

12.9 Button.Text

NAME
Button.Text – set/get button label

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

TYPE
String

APPLICABILITY
SG

12.10 Button.Toggle

NAME
Button.Toggle – create toggle button

FUNCTION
If you set this to `True`, a toggle button will be created. Toggle buttons 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

13 Checkbox class

13.1 Overview

Checkbox class derives from Area class and allows you to create checkbox widgets. A checkbox is a labelled box which by default is either on (checkmark is visible) or off (no checkmark). It is most commonly used to allow the user to toggle options. The XML tag's content is used as the checkbox label.

Here is an example of how to create a checkbox in XML:

```
<checkbox>Yes, I'm over 18 years old</checkbox>
```

13.2 Checkbox.Right

NAME
Checkbox.Right – define position of checkbox label

FUNCTION
Set this to `True` to have the checkbox image appear to the right of the accompanying label. By default, the checkbox image appears to the left of the label.

TYPE
Boolean

APPLICABILITY
I

13.3 Checkbox.Selected

NAME
Checkbox.Selected – set/get checkbox state

FUNCTION
Get and set the selected state of a checkbox object.

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

TYPE
Boolean

APPLICABILITY
ISGN

14 Choice class

14.1 Overview

Choice class derives from Area class and creates widgets that allow the user to select an entry from a predefined list of items. Only the selected item is visible until the user pulls down the menu of choices.

When you declare a choice widget, you can use the `<item>` tag to fill the choice widget with items. Here is an example XML excerpt for creating a choice widget with three entries:

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

Alternatively, you can also create an empty choice widget and fill it with entries later by using the `Choice.Insert` method. Here is how to create an empty choice widget:

```
<choice/>
```

14.2 Choice.Active

NAME

Choice.Active – set/get active choice item

FUNCTION

Set or get the active item in the choice widget ranging from index 0 for the first item to number of entries - 1 for the last item.

You can also set up a notification on this attribute to learn when the user changes the active choice item.

You can also pass the the special values `Next` or `Prev` to cycle through the choice widget's entries.

TYPE

Number or string (see above for possible values)

APPLICABILITY

ISGN

14.3 Choice.Clear

NAME

Choice.Clear – clear all entries (V1.1)

SYNOPSIS

```
moai.DoMethod(id, "Clear")
```

FUNCTION

Remove all entries from choice widget.

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

INPUTS

`id`	id of the choice object

14.4 Choice.Count

NAME

Choice.Count – get number of entries in choice widget (V1.1)

FUNCTION

Returns the current number of entries in the choice widget.

TYPE

Number

APPLICABILITY

G

14.5 Choice.GetEntry

NAME

Choice.GetEntry – get choice entry (V1.1)

SYNOPSIS

```
e$ = moai.DoMethod(id, "GetEntry", pos)
```

FUNCTION

Get an entry from a choice widget. You can pass either an absolute index in `pos` or the special value `Active` to get the active entry. `Choice.GetEntry` will then return the entry as a string.

INPUTS

`id`	id of the choice object
`pos`	entry index or "Active"

RESULTS

`e$`	choice entry at specified index

14.6 Choice.Insert

NAME

Choice.Insert – insert new entry (V1.1)

SYNOPSIS

```
moai.DoMethod(id, "Insert", pos, e$)
```

FUNCTION

Insert the entry specified in `e$` into the choice widget. 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 before the active entry. If there is no active entry, the entry will be inserted at the very top of the entry list.

`Bottom` Insert as last entry.

If `pos` is bigger or equal to the number of entries in the choice widget, the entry will be inserted as the last entry.

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

INPUTS

`id` id of the choice object

`pos` insert position as absolute number or special value (see above)

`e$` entry to insert

14.7 Choice.Remove

NAME

Choice.Remove – remove entry from choice widget (V1.1)

SYNOPSIS

```
moai.DoMethod(id, "Remove", pos)
```

FUNCTION

Remove an entry from a choice widget. 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.

When the active entry is removed, the following entry will become active.

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

INPUTS

`id` id of the choice object

`pos` index of entry to remove or one of the special values (see above)

14.8 Choice.Rename

NAME

Choice.Rename – rename an entry (V1.1)

SYNOPSIS

```
moai.DoMethod(id, "Rename", pos, newname$)
```

FUNCTION

Rename the choice entry at the specified position to the name specified in `newname$`. 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.

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

INPUTS

`id` id of the choice object

`pos` entry position as absolute number or special value (see above)

`newname$` new name for choice entry

15 Combobox class

15.1 Overview

Combobox class derives from Area class and creates a widget which is a combination of a list and a text widget. The user may either choose an entry from a predefined list of entries or he may enter individual data into the text widget.

When creating a combobox object, you can use the `<item>` tag to fill it with entries. Here is an example:

```
<combobox>
   <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>
</combobox>
```

You can also create an empty combobox and fill it with entries later by using the `Combobox.Insert` method. Here is how to create an empty combobox:

```
<combobox/>
```

15.2 Combobox.Acknowledge

NAME

Combobox.Acknowledge – learn when the user presses RETURN (V2.0)

FUNCTION

Whenever the user hits return this attribute will be set to `True`. You can listen to this notification and take the appropriate action.

TYPE

Boolean

APPLICABILITY

N

15.3 Combobox.Clear

NAME

Combobox.Clear – clear all entries (V1.1)

SYNOPSIS

```
moai.DoMethod(id, "Clear")
```

FUNCTION
Remove all entries from combobox.

INPUTS

`id`	id of the combobox object

15.4 Combobox.Close

NAME
Combobox.Close – close the combobox's list (V2.0)

SYNOPSIS
```
moai.DoMethod(id, "Close")
```

FUNCTION
Close the combobox's associated list of predefined values.

INPUTS

`id`	id of the combobox object

15.5 Combobox.Count

NAME
Combobox.Count – get number of entries in combobox (V1.1)

FUNCTION
Returns the current number of entries in the combobox.

TYPE
Number

APPLICABILITY
G

15.6 Combobox.GetEntry

NAME
Combobox.GetEntry – get combobox entry (V1.1)

SYNOPSIS
```
e$ = moai.DoMethod(id, "GetEntry", pos)
```

FUNCTION
Get entry at position `pos` from combobox widget.

INPUTS

`id`	id of the combobox object
`pos`	entry index

RESULTS

`e$` entry at specified index

15.7 Combobox.Insert

NAME

Combobox.Insert – insert new entry (V1.1)

SYNOPSIS

```
moai.DoMethod(id, "Insert", pos, e$)
```

FUNCTION

Insert the entry specified in `e$` into the combobox. 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.

`Bottom` Insert as last entry.

If `pos` is bigger or equal to the number of entries in the combobox, the entry will be inserted as the last entry.

INPUTS

`id` id of the combobox object

`pos` insert position as absolute number or special value (see above)

`e$` entry to insert

15.8 Combobox.Open

NAME

Combobox.Open – open the combobox's list (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "Open")
```

FUNCTION

Pop up the combobox's associated list of predefined values.

INPUTS

`id` id of the combobox object

15.9 Combobox.Popup

NAME
Combobox.Popup – learn about open and close events (V2.0)

FUNCTION
Whenever the user opens or closes the combobox's list, this attribute will trigger a notification. The `TriggerValue` will contain the current state of the popup list, i.e. `True` for open and `False` for closed.

TYPE
Boolean

APPLICABILITY
N

15.10 Combobox.Remove

NAME
Combobox.Remove – remove entry from combobox (V1.1)

SYNOPSIS
```
moai.DoMethod(id, "Remove", pos)
```

FUNCTION
Remove entry at the position specified by `pos` from combobox.

INPUTS

`id`	id of the combobox object
`pos`	index of entry to remove

15.11 Combobox.Rename

NAME
Combobox.Rename – rename an entry (V1.1)

SYNOPSIS
```
moai.DoMethod(id, "Rename", pos, newname$)
```

FUNCTION
Rename the combobox entry at the specified position to the name specified in `newname$`. The entry position is specified in the `pos` argument. This must be an absolute index position starting at 0 for the first entry.

INPUTS

`id`	id of the combobox object
`pos`	entry position as absolute number
`newname$`	new name for combobox entry

15.12 Combobox.Selected

NAME
Combobox.Selected – learn about selection changes (V2.0)

FUNCTION
Whenever the user selects an item from the combobox's list, this attribute will trigger.

TYPE
Boolean

APPLICABILITY
N

15.13 Combobox.Value

NAME
Combobox.Value – set/get current combobox contents

FUNCTION
Get and set the current combobox contents.

You can also set up a notification on this attribute to be notified whenever the contents of the combobox object change.

TYPE
String

APPLICABILITY
ISGN

16 Dialog class

16.1 Overview

Dialog class derives from Window class and creates modal (or modeless) dialogs. Modal dialogs are special top-level windows which block the rest of the application until they are closed. Modal dialogs are typically used when user action is required to continue with the program or to indicate that the application is currently busy. A dialog could show a progress bar then, for example.

As with windows, the root element of a dialog always needs to be a single group object, i.e. an instance of Group class. See Section 18.1 [Group class], page 105, for details. Additionally, it is not allowed to have multiple elements at the dialog's root level. You must only use a single group object as the root element. Here's an example of a simple progress dialog in XML:

```
<dialog id="mydlg" title="Working...">
   <vgroup>
      <progressbar id="prg"/>
      <button id="cancel">Cancel</button>
   </vgroup>
</dialog>
```

You should create dialogs only when you need them and destroy them as soon as you are finished with them. It is not advised to create all your dialogs on startup using `moai.CreateApp()` and keep them in memory all the time. Instead, you should use `moai.CreateDialog()` to create a dialog when you need it and then have it destroyed as soon as you are finished with it. The reason for this is that windows are quite a finite resource on some operating systems supported by RapaGUI. For example, on Windows there is a limit of about 10,000 windows per process. That might sound like sufficiently enough but keep in mind that on Windows every widget is a "window" for the operating system, e.g. every label, button, frame, checkbox, group, etc. in your application is a window so you should take care that you create your dialogs only as needed using `moai.CreateDialog()` and destroy them right afterwards.

In practice, it is advised to create a separate XML file for each of your dialogs and then use `moai.CreateDialog()` to convert the XML file into a dialog at runtime and have the dialog destroyed automatically by RapaGUI as soon as the user closes it.

As a subclass of Window class most Window class attributes and methods can be used with Dialog class as well. See Section 62.1 [Window class], page 301, for details. But note that modal dialogs mustn't be opened by setting `Window.Open` but by running the `Dialog.ShowModal` method. To close a modal dialog, simply call the `Dialog.EndModal` method. This will also automatically destroy the dialog, i.e. it will implicitly call `moai.FreeDialog()` on the dialog unless you explicitly request that the dialog should not be destroyed by setting an optional argument to `True`.

Similarly, the dialog will also be automatically destroyed when the user clicks the close box of a dialog. If you don't want this or if you need to customize the application's behaviour when clicking the close box of a dialog, you have to install a listener on the

`Window.CloseRequest` attribute. If there is no listener on this attribute, RapaGUI will simply call `Dialog.EndModal` with 0 as the parameter when the dialog's close box is clicked.

If you want to open a modeless dialog, i.e. a dialog that doesn't block your script while it is open, you must not use `Dialog.ShowModal` but set `Window.Open` to `True` for the dialog instead (and possibly set `Application.Sleep` to `True` too, if you wish to put the application in some sort of sleep state while the dialog is open). You can then implement the dialog's actual behaviour (e.g. a progress bar that is constantly being updated) by repeatedly calling a management function either using `SetTimeout()` or Hollywood 9.0's new `RunCallback()` function. When you're finished with a modeless dialog, just set `Window.Open` to `False` on it. Do not use `Dialog.EndModal` with modeless dialogs! See the `"Dialogs"` example that comes with RapaGUI for an example of a modeless dialog (choose `"Test progress bar dialog"` in the menu).

16.2 Dialog.EndModal

NAME

Dialog.EndModal – close dialog and end modal loop

SYNOPSIS

```
moai.DoMethod(id, "EndModal", retval[, nodestroy])
```

FUNCTION

This method will close the specified dialog that has been opened using `Dialog.ShowModal` and it will break its modal loop. The value you pass in `retval` will be the return value of the call to this dialog's `Dialog.ShowModal` method then. This value is often used to indicate success or failure, i.e. you could return `True` in case the user pressed the `"OK"` button, and `False` otherwise.

Additionally, this method will automatically destroy the dialog, i.e. it will implicitly call `moai.FreeDialog()` on the dialog object. If you don't want this, you have to set the optional argument `nodestroy` to `True`. However, it is recommended and it is good programming practice to destroy every dialog as soon as you are finished with it. See Section 16.1 [Dialog class], page 95, for details.

INPUTS

`id` id of the dialog object

`retval` desired return value for `Dialog.ShowModal`

`nodestroy` optional: `True` if this method shouldn't automatically destroy the dialog (defaults to `False` which means destroy the dialog after closing it)

16.3 Dialog.ShowModal

NAME

Dialog.ShowModal – open dialog and begin modal loop

SYNOPSIS

```
retval = moai.DoMethod(id, "ShowModal")
```

FUNCTION

This method opens the dialog window and begins a modal loop that blocks the rest of the application and waits for the dialog to be closed before returning control to the script. In other words, `Dialog.ShowModal` will not return before the user either closes the dialog or the script calls `Dialog.EndModal` on the dialog, whatever happens first.

If you want to open a modeless dialog, i.e. a dialog that doesn't block your script while it is open, you must not use `Dialog.ShowModal` but set `Window.Open` to `True` for the dialog instead (and possibly set `Application.Sleep` to `True` too, if you wish to put the application in some sort of sleep state while the dialog is open). You can then implement the dialog's actual behaviour (e.g. a progress bar that is constantly being updated) by repeatedly calling a management function either using `SetTimeout()` or Hollywood 9.0's new `RunCallback()` function. When you're finished with a modeless dialog, just set `Window.Open` to `False` on it. Do not use `Dialog.EndModal` with modeless dialogs!

Note that it is good programming practice to create and destroy dialogs as needed. You should not keep them in memory for the complete lifetime of your application but only create them when necessary and destroy them as soon as you are finished with them. See Section 16.1 [Dialog class], page 95, for details.

If you want to be notified when the user clicks the close box of a dialog, you have to install a listener on the `Window.CloseRequest` attribute. If there is no listener on this attribute, RapaGUI will simply call `Dialog.EndModal` with 0 as the parameter when the dialog's close box is clicked.

INPUTS

`id`	id of the dialog object

RESULTS

`retval`	dialog return value

EXAMPLE

See Section 6.3 [moai.CreateObject], page 47.

17 Finddialog class

17.1 Overview

Finddialog class derives from Dialog class and creates a standard system find dialog. Find dialogs are typically used in connection with Texteditor class to allow the user to search through the entire contents of a texteditor widget.

Even though Finddialog class is a subclass of Dialog class, the dialogs it creates must always be modeless. Thus, it is neither allowed to show them using `Dialog.ShowModal` nor is it allowed to close them using `Dialog.EndModal`. Instead, dialogs created by Finddialog class must always be shown and hidden by setting the `Window.Open` attribute.

Finddialog class is available since RapaGUI 2.0.

Note that this class is currently unsupported on AmigaOS (and compatibles) and Android.

17.2 Finddialog.Down

NAME
Finddialog.Down – set/get search direction (V2.0)

FUNCTION
Set/get the search direction (`True` for searching downwards, `False` for searching upwards).

TYPE
Boolean

APPLICABILITY
ISG

17.3 Finddialog.Find

NAME
Finddialog.Find – learn when the find button is pressed (V2.0)

FUNCTION
Set up a notification on this attribute to learn when the dialog's find button is pressed.

TYPE
Boolean

APPLICABILITY
N

17.4 Finddialog.FindNext

NAME
Finddialog.FindNext – learn when the find next button is pressed (V2.0)

FUNCTION
Set up a notification on this attribute to learn when the dialog's find next button is pressed.

TYPE
Boolean

APPLICABILITY
N

17.5 Finddialog.FindString

NAME
Finddialog.FindString – set/get current find string (V2.0)

FUNCTION
Set/get the current find string.

TYPE
String

APPLICABILITY
ISG

17.6 Finddialog.MatchCase

NAME
Finddialog.MatchCase – set/get case sensitivity setting (V2.0)

FUNCTION
Set/get the case sensitivity setting in dialog.

TYPE
Boolean

APPLICABILITY
ISG

17.7 Finddialog.NoMatchCase

NAME
Finddialog.NoMatchCase – use case-insensitive search (V2.0)

FUNCTION
If you set this attribute to `True`, the dialog won't allow toggling case sensitive searching.

TYPE
Boolean

APPLICABILITY
I

17.8 Finddialog.NoUpDown

NAME
Finddialog.NoUpDown – forbid changing the search direction (V2.0)

FUNCTION
If you set this attribute to `True`, the dialog won't allow changing the search direction.

TYPE
Boolean

APPLICABILITY
I

17.9 Finddialog.NoWholeWord

NAME
Finddialog.NoWholeWord – forbid searching for whole words (V2.0)

FUNCTION
If you set this attribute to `True`, the dialog won't allow searching for whole words.

TYPE
Boolean

APPLICABILITY
I

17.10 Finddialog.Replace

NAME
Finddialog.Replace – learn when the replace button is pressed (V2.0)

FUNCTION
Set up a notification on this attribute to learn when the dialog's replace button is pressed.

TYPE
Boolean

APPLICABILITY
N

17.11 Finddialog.ReplaceAll

NAME
Finddialog.ReplaceAll – learn when the replace all button is pressed (V2.0)

FUNCTION
Set up a notification on this attribute to learn when the dialog's replace all button is pressed.

TYPE
Boolean

APPLICABILITY
N

17.12 Finddialog.ReplaceMode

NAME
Finddialog.ReplaceMode – put finddialog into replace mode (V2.0)

FUNCTION
Create a replace dialog instead of a search dialog.

TYPE
Boolean

APPLICABILITY
I

17.13 Finddialog.ReplaceString

NAME
Finddialog.ReplaceString – set/get current replace string (V2.0)

FUNCTION
Set/get the current replace string.

TYPE
String

APPLICABILITY
ISG

17.14 Finddialog.WholeWord

NAME
Finddialog.WholeWord – set/get whole word searching (V2.0)

FUNCTION
Set/get the whole word searching flag in the find dialog.

TYPE
Boolean

APPLICABILITY
ISG

18 Group class

18.1 Overview

Group class can be used to lay out a number of children in various ways. When the parent's size changes, groups relayout their children which can be very useful to create windows that are freely resizable in all directions.

The following different group types are supported by RapaGUI:

`<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.

`<scrollgroup>`
: A group with scrollbars. This is a special group which shows an embedded group with scrollbars. See Section 44.1 [Scrollgroup class], page 221, for details.

Column groups are useful if you need to have identical widget sizes for all your children in a group for a more pleasant visual appearance. For example, imagine a form made up of text entry widgets 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">
   <label>Name</label>
   <textentry/>
   <label>Street</label>
   <textentry/>
   <label>City</label>
   <textentry/>
   <label>Zip code</label>
   <textentry/>
   <label>Country</label>
   <textentry/>
   <label>Telephone</label>
   <textentry/>
   <label>Email</label>
   <textentry/>
</colgroup>
```

If we used a `<vgroup>` with one `<hgroup>` per row the appearance would be pretty bad because the text entry widget's width would be different for each line which looks pretty ugly.

Note that Group class doesn't derive from Area class because groups do not exist as physical widgets but they are just layout tools for their child widgets or groups. This is why you cannot use attributes and methods from Area class on group objects.

18.2 Group.Append

NAME

Group.Append – add detached object as last group child

SYNOPSIS

```
moai.DoMethod(id, "Append", 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 removal of children. This can be done by running the `Group.InitChange` and `Group.ExitChange` methods on the respective group object.

Detached MOAI objects can be created either by calling the `moai.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 6.3 [moai.CreateObject], page 47.

18.3 Group.Color

NAME

Group.Color – set group background color

FUNCTION

Set the background color for the group. This must only be used with `<hgroup>` or `<vgroup>` objects. It isn't supported for `<colgroup>` objects. Also, you must not set `Group.Frame` to `True` when using this attribute. If you want to set the background color of a framed group, you have to create a helper group around the group whose background color you want to set, e.g.

```
<vgroup frame="true" padding="0">
    <vgroup color="#ffffff">
        ...
    </vgroup>
</vgroup>
```

If you want to set the background color for a `<colgroup>`, you can also just create a helper group around the column group (see above).

TYPE

Number

APPLICABILITY
I

18.4 Group.Columns

NAME
Group.Columns – define group columns

FUNCTION
Set the number of columns for a two dimensional grid group. If you specify this tag, make sure that the grid is balanced, i.e. there is no remainder when dividing the total number of children by the number of columns.

This attribute must only be used with the `<colgroup>` tag.

TYPE
Number

APPLICABILITY
I

18.5 Group.ExitChange

NAME
Group.ExitChange – terminate group exchange state

SYNOPSIS
```
moai.DoMethod(id, "ExitChange"[, force])
```

FUNCTION
This method terminates the state established by `Group.InitChange`. If children have been added or removed, RapaGUI will refresh the group making the changes visible to the user. You can force RapaGUI to do this refresh by setting the "force" argument to `True`. In that case RapaGUI 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`	optional: specify `True` here to force a complete refresh; otherwise RapaGUI will only refresh the group if objects have been added or removed

EXAMPLE
See Section 6.3 [moai.CreateObject], page 47.

18.6 Group.Frame

NAME
Group.Frame – create framed group

FUNCTION
Set this attribute to `True` to add a frame around this group. You can also add a title for the frame by setting the `Group.FrameTitle` attribute.

TYPE
Boolean

APPLICABILITY
I

18.7 Group.FrameTitle

NAME
Group.FrameTitle – set title for framed group

FUNCTION
If you are creating a framed group by setting `Group.Frame` to `True`, you can use this attribute to add a title text for the frame. This attribute must only be used if `Group.Frame` has been set to `True`.

Note that changing a frame's title requires MUI 4.0 or better on AmigaOS and compatibles.

TYPE
String

APPLICABILITY
ISG

18.8 Group.HAlign

NAME
Group.HAlign – set default horizontal alignment

FUNCTION
Set this attribute to define how non-resizable children should be aligned in case they are smaller than the size allocated for them. The following options are available:

`Left`	Left alignment.
`Right`	Right alignment.
`Center`	Centered alignment. This is the default.

TYPE
String (see above for possible values)

APPLICABILITY
I

18.9 Group.Hide

NAME
Group.Hide – show/hide group

FUNCTION
Set this attribute to show or hide a group. Note that groups maintain their private visibility state, so setting this flag doesn't mean that the group simply sets `Area.Hide` on all its children. Both, widgets and groups, maintain their own visibility state.

TYPE
Boolean

APPLICABILITY
ISG

18.10 Group.HorizSpacing

NAME
Group.HorizSpacing – set horizontal spacing

FUNCTION
Set the number of device-independent pixels to use as horizontal spacing between the group's children.

TYPE
Number

APPLICABILITY
I

18.11 Group.Icon

NAME
Group.Icon – set group icon

FUNCTION
If you define groups for a pageview, you can use this attribute to specify an image for this group that shall be displayed in the widget that is used to browse through the individual pages. See Section 34.1 [Pageview class], page 189, for details. Whether this attribute expects a Hollywood brush or icon, depends on what you specify in the `Group.IconType` attribute. By default, `Group.Icon` expects a Hollywood brush. To remove the image from a group page, set this attribute to -1.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

Starting with RapaGUI 1.1 this attribute has an applicability of `ISG`. Before that, its applicability was just `I`. Note that on AmigaOS and compatibles changing the group icon at runtime for all pageview modes except list mode is only available on MUI 4.0 or higher.

TYPE
Number

APPLICABILITY
ISG

18.12 Group.IconScale

NAME
Group.IconScale – configure automatic image scaling (V2.0)

FUNCTION
If `Group.Icon` has been set to a raster brush and RapaGUI is running on a high-DPI display, RapaGUI will automatically scale the brush's raster graphics to fit to the current monitor's DPI setting. If you don't want that, set this tag to `False`.

Alternatively, you can also globally disable automatic image scaling by setting the `ScaleGUI` tag to `False` when calling `@REQUIRE` on RapaGUI. See Section 3.4 [Initializing RapaGUI], page 11, for details.

Please also read the chapter about high-DPI support in RapaGUI to learn more about supporting high-DPI displays. See Section 3.12 [High-DPI support], page 20, for details.

TYPE
Boolean

APPLICABILITY
I

18.13 Group.IconType

NAME
Group.IconType – set icon type to use (V2.0)

FUNCTION
This attribute allows you to set the type of the Hollywood image object passed in the `Group.Icon` attribute. By default, `Group.Icon` expects a Hollywood brush. By setting `Group.IconType`, however, you can make it use a different Hollywood image type.

The following image types are currently available:

`Brush` Use a Hollywood brush. This is the default type. You can use either raster or vector brushes. Vector brushes have the advantage that they can be scaled to any resolution without losses in quality. This is very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

`Icon` Use a Hollywood icon. This image type has the advantage that it can contain several subimages of different sizes. This makes it possible to provide images in different resolutions which can be very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

Note that you can globally change the default of all `IconType` attributes to Hollywood icons by setting the `Application.UseIcons` tag. See Section 9.13 [Application.UseIcons], page 65, for details.

TYPE
String (see above for possible values)

APPLICABILITY
I

18.14 Group.InitChange

NAME
Group.InitChange – prepare group for addition or removal of children

SYNOPSIS
```
moai.DoMethod(id, "InitChange")
```

FUNCTION
Prepare a group for addition or removal of children. Whenever you call methods like `Group.Append`, `Group.Prepend`, `Group.Insert` or `Group.Remove` you first have to call this method to put the group into a special exchange state which allows the addition and removal of children. When you're finished, use `Group.ExitChange` to relayout the group.

INPUTS

`id` id of the group object

EXAMPLE
See Section 6.3 [moai.CreateObject], page 47.

18.15 Group.Insert

NAME
Group.Insert – insert detached object after specified child

SYNOPSIS

```
moai.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 removal of children. This can be done by running the `Group.InitChange` and `Group.ExitChange` methods on the respective group object.

Detached MOAI objects can be created either by calling the `moai.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 6.3 [moai.CreateObject], page 47.

18.16 Group.Padding

NAME

Group.Padding – set padding space between frame and group

FUNCTION

When creating a framed group, you can use this attribute to define the number of padding device-independent pixels between the frame and the group's children.

Additionally, on Windows, Linux, and macOS this attribute also defines the number of padding pixels around the group when the group is part of a pageview object. See Section 34.1 [Pageview class], page 189, for details.

TYPE

Number

APPLICABILITY

I

18.17 Group.Paint

NAME

Group.Paint – request paint notification

FUNCTION

Set up a listener on this attribute to have your event handler called whenever the group background needs to be drawn. Your event handler can then draw custom graphics to the group background.

RapaGUI will pass the identifier of a Hollywood brush whose size is exactly as big as the group background. You then have to draw the desired background graphics to this brush. Precisely, you just have to draw to the rectangle defined by the four coordinates `X`, `Y`, `Width`, and `Height` which are passed to your callback as well. These four coordinates describe a rectangular area within the dimensions of the brush that is passed to your callback. When a full redraw is needed, `X` and `Y` will be 0 and `Width` and `Height` will match the dimensions of the brush. Most oftenly, however, only a partial redraw is needed and then you must only draw to the portion of the brush defined by those coordinates.

The following extra arguments will be passed to your event handler:

`Brush:` Contains the identifier of a brush you have to draw to. Use Hollywood's `SelectBrush()` command to select this brush as the output device in your callback. Don't forget to call `EndSelect()` when you are done!

`ViewWidth:` Contains the total group width. This is also identical to the width of the brush that is passed to your callback.

`ViewHeight:` Contains the total group height. This is also identical to the height of the brush that is passed to your callback.

`X:` Contains the x-position inside the brush at which you should start drawing. See above for details.

`Y:` Contains the y-position inside the brush at which you should start drawing. See above for details.

`Width:` Contains the number of columns you should paint to the brush (starting from `X`). See above for details.

`Height:` Contains the number of rows you should paint to the brush (starting from `Y`). See above for details.

Note that `Group.Paint` must only be used with `<hgroup>` or `<vgroup>` objects. It isn't supported for `<colgroup>` objects. Also, you must not set `Group.Frame` to `True` when using this attribute. If you want to draw the background of a framed group, you have to create a helper group around the group whose background you want to draw, e.g.

```
<vgroup frame="true" padding="0">
    <vgroup notify="paint">
        ...
    </vgroup>
</vgroup>
```

If you want to draw the background of a `<colgroup>`, you can also just create a helper group around the column group (see above).

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

TYPE
Boolean

APPLICABILITY
N

18.18 Group.Prepend

NAME
Group.Prepend – add detached object as first group child

SYNOPSIS
`moai.DoMethod(id, "Prepend", 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 removal of children. This can be done by running the `Group.InitChange` and `Group.ExitChange` methods on the respective group object.

Detached MOAI objects can be created either by calling the `moai.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 6.3 [moai.CreateObject], page 47.

18.19 Group.Remove

NAME
Group.Remove – detach object from group

SYNOPSIS
`moai.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 `moai.FreeObject()`.

Before you can call this method, you have to put the group into a special state that allows the addition and removal 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

```
moai.DoMethod("mygroup", "initchange")
moai.DoMethod("mygroup", "remove", "mychild")
moai.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.

18.20 Group.SameSize

NAME

Group.SameSize – set same size for all children

FUNCTION

Set this attribute to `True` to force the group to use the same size for all children. This often creates a more consistent and better look. RapaGUI automatically sets this attribute for horizontal groups which contain a row of buttons.

TYPE

Boolean

APPLICABILITY

I

18.21 Group.Spacing

NAME

Group.Spacing – set spacing

FUNCTION

Set the number of device-independent pixels to use as horizontal and vertical spacing between the group's children. Setting this attribute has the same effect as setting both `Group.HorizSpacing` and `Group.VertSpacing`.

TYPE

Number

APPLICABILITY

I

18.22 Group.Title

NAME
Group.Title – set group title

FUNCTION
If you define groups for a pageview, you can use this attribute to specify a title for this group that shall be displayed in the widget that is used to browse through the individual pages. See Section 34.1 [Pageview class], page 189, for details.

Starting with RapaGUI 1.1 this attribute has an applicability of `ISG`. Before that, its applicability was just `I`. Note that on AmigaOS and compatibles changing the group title at runtime for all pageview modes except list mode is only available on MUI 4.0 or higher.

TYPE
String

APPLICABILITY
ISG

18.23 Group.VAlign

NAME
Group.VAlign – set default vertical alignment

FUNCTION
Set this attribute to define how non-resizable children should be aligned in case they are smaller than the size allocated for them. The following options are available:

`Top`	Top alignment.
`Bottom`	Bottom alignment.
`Center`	Centered alignment. This is the default.

TYPE
String (see above for possible values)

APPLICABILITY
I

18.24 Group.VertSpacing

NAME
Group.VertSpacing – set vertical spacing

FUNCTION
Set the number of device-indepdent pixels to use as vertical spacing between the group's children.

TYPE
Number

APPLICABILITY
I

18.25 Group.Weight

NAME
Group.Weight – set group weight

FUNCTION
This attribute controls the weight of a group object inside another group of objects. By default, all objects in a group have a weight of 100. If you want to have an object that appears twice as large, you have to give it a weight of 200. To make a group appear one and a half times as large, just use 150 and so on.

Naturally, this attribute is only useful for groups which are resizable.

TYPE
Number

APPLICABILITY
I

19 HLine class

19.1 Overview

HLine class derives from Area class and creates a horizontal divider line which can be used separate groups of widgets. There is also VLine class which creates vertical divider lines. See Section 59.1 [VLine class], page 293, for details.

HLine class doesn't define any attributes.

20 Hollywood class

20.1 Overview

Hollywood class derives from Area class and allows you to embed a complete Hollywood display inside your GUI as a widget. Whenever you draw something to a Hollywood display that is attached to a widget, it will automatically be drawn into your widget 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 widget as normal Hollywood events. Thus, Hollywood class allows you to transparently use almost all of Hollywood's powerful features inside a widgets.

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

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

Note that by default Hollywood widgets are not resizable. You can change this by setting the `Area.FixWidth` and `Area.FixHeight` attributes accordingly. If you set one of those attributes to `False`, you will get a resizable Hollywood widget and just like with normal Hollywood displays, your Hollywood widget will also be sent a `SizeWindow` message whenever the widget size changes. You can then adapt your widget's content to the new size.

Also note that by default, all Hollywood widgets might be scaled to fit to the current monitor's DPI setting. If you don't want that, you can set the `ScaleHollywood` tag in `@REQUIRE` to `False`. See Section 3.12 [High-DPI support], page 20, for details.

See Section 3.19 [Hollywood bridge], page 26, for details.

20.2 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.

By default, the widget won't be resizable. You can change this by setting the `Area.FixWidth` and `Area.FixHeight` attributes to `False`. In that case, your Hollywood widget will become resizable and whenever the user resizes the window, your Hollywood display will get a `"SizeWindow"` event which you can listen to using the `InstallEventHandler()` function. You can then react on this event accordingly and redraw your display, etc.

TYPE

Number

APPLICABILITY

IS

20.3 Hollywood.DropFile

NAME
Hollywood.DropFile – learn about dropped files (V1.1)

FUNCTION
When setting up a notification on this attribute, RapaGUI will run your event callback whenever one or more files have been dropped onto the Hollywood widget. The `TriggerValue` field of the message table will be set to a table that contains a list of all files that have been dropped onto the widget.

Additionally, the message table will contain the following two extra fields:

`X:` The x-position where the user has dropped the file(s). This will be relative to the widget's left corner.

`Y:` The y-position where the user has dropped the file(s). This will be relative to the widget's top corner.

See Section 3.7 [Notifications], page 13, for details.

Note that `Hollywood.DropFile` is only ever triggered if `Hollywood.DropTarget` has been set to `True` first.

TYPE
Boolean

APPLICABILITY
N

20.4 Hollywood.DropTarget

NAME
Hollywood.DropTarget – configure drop target settings (V1.1)

FUNCTION
Set this to `True` if files can be dropped on this Hollywood widget. You can listen to the `Hollywood.DropFile` attribute to learn when the user drops one or more files on the Hollywood widget.

TYPE
Boolean

APPLICABILITY
ISG

21 HSpace class

21.1 Overview

HSpace class derives from Area class and creates objects of a fixed device-independent pixel size. This is typically used to fine-tune the GUI layout. To create padding objects that are freely resizable, you can use Rectangle class instead. See Section 41.1 [Rectangle class], page 209, for details.

21.2 HSpace.Width

NAME
HSpace.Width – set horizontal space

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

TYPE
Number

APPLICABILITY
I

22 HSplitter class

22.1 Overview

HSplitter class is a special variant of Group class. It creates a horizontal group of two children with a sash between them that allows the user to individually adjust the sizes of the group's two children by dragging the sash.

Note that HSplitter groups must always contain exactly two children. Additionally, the children must also be sizable because HSplitter class allows its children to be resized. Here is an example XML excerpt for creating a horizontal splitter layout with two textviews:

```
<hsplitter>
    <textview>One</textview>
    <textview>Two</textview>
</hsplitter>
```

HSplitter class is available since RapaGUI 2.0.

22.2 HSplitter.Border

NAME
HSplitter.Border – set sash border style (V2.0)

FUNCTION
Set this to `False` to make the sash use a different border style. Note that this is not supported on all platforms.

TYPE
Boolean

APPLICABILITY
I

22.3 HSplitter.Gravity

NAME
HSplitter.Gravity – set/get sash gravity (V2.0)

FUNCTION
Set or get the sash gravity. Gravity is a value between 0 and 100 which controls the position of the sash while resizing the splitter group. The gravity value tells splitter group how much the left child will grow while resizing. For example:

- 0: Only the right child is automatically resized.
- 50: Both children grow by equal size.
- 100: Only the left child grows.

The default sash gravity is 0.

TYPE
Number

APPLICABILITY
SG

22.4 HSplitter.MinPaneSize

NAME
HSplitter.MinPaneSize – set/get minimum pane size (V2.0)

FUNCTION
Set or get the minimum pane size. The default minimum pane size is 0, which means that either pane can be reduced to zero by dragging the sash, thus removing one of the panes. To prevent this behaviour (and veto out-of-range sash dragging), set a minimum size, for example 20.

This value is in device-independent pixels.

TYPE
Number

APPLICABILITY
ISG

22.5 HSplitter.Position

NAME
HSplitter.Position – set/get sash position (V2.0)

FUNCTION
Set or get the position of the sash dividing the group's children. This value is in device-independent pixels.

TYPE
Number

APPLICABILITY
SG

22.6 HSplitter.Split

NAME
HSplitter.Split – split the panes (V2.0)

SYNOPSIS
```
moai.DoMethod(id, "Split", pos)
```

FUNCTION
This splits the group into two panes. The `pos` argument specifies the position of the sash in device-independent pixels. If this value is positive, it specifies the size of the left pane. If it is negative, its absolute value gives the size of the right pane. Finally, specify 0 to choose the default position (half of the total window width).

INPUTS

`id`	id of the splitter object
`pos`	desired sash position

22.7 HSplitter.Unsplit

NAME

HSplitter.Unsplit – unsplit the panes (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "Unsplit", idx)
```

FUNCTION

This unsplits the panes and hides the pane specified in `idx`. If `idx` is 0, the left pane will be hidden, if it is 1, the right pane will be hidden. To make both panes visible again, use the `HSplitter.Split` method.

INPUTS

`id`	id of the splitter object
`idx`	index of pane to remove

23 HTMLview class

23.1 Overview

HTMLview class derives from Area class and creates a widget which displays a HTML page either provided as raw data or as a file source stored locally or on a remote server. It also supports some browser features like a history of visited pages and the ability to step through this history.

On Windows and macOS this class uses an HTML control provided by the OS. On Linux this class requires WebKitGTK+ to be installed. Since this is not available on every system, there is also a separate RapaGUI version without this class and thus without a WebKitGTK+ dependency. On AmigaOS and compatibles this class requires the `HTMLview.mcc` extension.

23.2 HTMLview.CanGoBack

NAME
HTMLview.CanGoBack – learn if browser can go backward

FUNCTION
This attribute is set to `True` whenever there is a page in the history that the browser can go back to. You can set up a notification on this attribute and disable/enable the "Back" button of your toolbar depending on the state of this attribute.

TYPE
Boolean

APPLICABILITY
GN

23.3 HTMLview.CanGoForward

NAME
HTMLview.CanGoForward – learn if browser can go forward

FUNCTION
This attribute is set to `True` whenever there is a page in the history that the browser can go forward to. You can set up a notification on this attribute and disable/enable the "Forward" button of your toolbar depending on the state of this attribute.

TYPE
Boolean

APPLICABILITY
GN

23.4 HTMLview.ClearHistory

NAME
HTMLview.ClearHistory – clear history

SYNOPSIS
`moai.DoMethod(id, "ClearHistory")`

FUNCTION
Clears the browser history.

INPUTS

`id`	id of the HTMLview object

23.5 HTMLview.Contents

NAME
HTMLview.Contents – set/get widget contents

FUNCTION
Set or get the HTML data displayed by the widget. If you set this attribute, you have to pass valid HTML formatted code to the widget. Getting this attribute allows you to obtain the source code of the page currently viewed.

TYPE
String

APPLICABILITY
ISG

23.6 HTMLview.File

NAME
HTMLview.File – set/get file to show

FUNCTION
Set the file to be shown in the widget. This is usually a HTML page but you could also pass a path to an image.

Note that in contrast to `HTMLview.URL` this attribute expects just a normal path to a file, i.e. don't use the `file://` protocol prefix here.

TYPE
String

APPLICABILITY
ISG

23.7 HTMLview.GoBack

NAME
HTMLview.GoBack – step backwards through the history

SYNOPSIS
```
moai.DoMethod(id, "GoBack")
```

FUNCTION
Navigates back in the history of visited pages.

INPUTS

`id`	id of the HTMLview object

23.8 HTMLview.GoForward

NAME
HTMLview.GoForward – step forward through the history

SYNOPSIS
```
moai.DoMethod(id, "GoForward")
```

FUNCTION
Navigates forward in the history of visited pages.

INPUTS

`id`	id of the HTMLview object

23.9 HTMLview.Reload

NAME
HTMLview.Reload – reload current page

SYNOPSIS
```
moai.DoMethod(id, "Reload")
```

FUNCTION
Reloads the current page.

INPUTS

`id`	id of the HTMLview object

23.10 HTMLview.Search

NAME
HTMLview.Search – search current page

SYNOPSIS
```
found = moai.DoMethod(id, "Search", t$, flags$)
```

FUNCTION
Search the current page for `t$` and if found, scroll the result into view and select it.
`flags$` can be a combination of the following flags:

`CaseSensitive`
: Search in a case sensitive manner.

`Backwards`
: Search in reverse direction.

If you specify multiple options in `flags$`, separate them using a semicolon.

INPUTS

`id`	id of the HTMLview object
`t$`	string to search for
`flags$`	combination of flags or empty string for default options

RESULTS

`found`	returns `True` or `False`, depending on whether the text was found (V2.0)

23.11 HTMLview.Title

NAME
HTMLview.Title – get title of current page

FUNCTION
Gets the title of the page currently viewed, i.e. the value of the `<title>` tag.

TYPE
String

APPLICABILITY
G

23.12 HTMLview.URL

NAME
HTMLview.URL – set/get current URL

FUNCTION
Set the URL to be shown in the widget. This URL must include the protocol prefix, typically `http://`. When opening local files you have to use `file://` as a protocol but you could also just use `HTMLview.File` instead which is more convenient in this case.

TYPE
String

APPLICABILITY
ISG

24 Hyperlink class

24.1 Overview

Hyperlink class derives from Area class and creates a clickable text object that opens the specified URL in a browser when the user clicks on it. Here is an example XML excerpt for creating a hyperlink widget that will open http://www.hollywood-mal.com when you click on it:

```
<hyperlink>http://www.hollywood-mal.com</hyperlink>
```

If you don't want the URL to be shown, you can use the `Hyperlink.Label` attribute to set a different label for the hyperlink object, for example:

```
<hyperlink label="Click me">http://www.hollywood-mal.com</hyperlink>
```

Hyperlink class is available since RapaGUI 2.0.

24.2 Hyperlink.Label

NAME
Hyperlink.Label – set/get label text (V2.0)

FUNCTION
Set or get the label text to be shown by the hyperlink widget. If this is not specified, the hyperlink widget will show the URL that has been set using `Hyperlink.URL`.

TYPE
String

APPLICABILITY
ISG

24.3 Hyperlink.URL

NAME
Hyperlink.URL – set/get target URL (V2.0)

FUNCTION
Set or get the target URL that should be opened in a browser when the user clicks on the hyperlink widget. This must always be provided.

TYPE
String

APPLICABILITY
SG

25 Image class

25.1 Overview

Image class derives from Area class and shows the specified image. The image pixel data is taken from a Hollywood brush or icon, depending on what is set in `Image.BrushType`.

Here is an example XML declaration:

```
<image 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/icons so you can also embed images with transparent areas.

25.2 Image.Brush

NAME

Image.Brush – set image to display

FUNCTION

Set this attribute to the identifier of a Hollywood brush or icon to specify the image that should be shown by the widget. Whether this attribute expects a Hollywood brush or icon, depends on what you specify in the `Image.BrushType` attribute. As the name implies, `Image.Brush` expects a Hollywood brush by default.

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

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

TYPE

Number

APPLICABILITY

IS

25.3 Image.BrushScale

NAME

Image.BrushScale – configure automatic image scaling (V2.0)

FUNCTION

If `Image.Brush` has been set to a raster brush and RapaGUI is running on a high-DPI display, RapaGUI will automatically scale the brush's raster graphics to fit to the current monitor's DPI setting. If you don't want that, set this tag to `False`.

Alternatively, you can also globally disable automatic image scaling by setting the `ScaleGUI` tag to `False` when calling `@REQUIRE` on RapaGUI. See Section 3.4 [Initializing RapaGUI], page 11, for details.

Please also read the chapter about high-DPI support in RapaGUI to learn more about supporting high-DPI displays. See Section 3.12 [High-DPI support], page 20, for details.

TYPE
Boolean

APPLICABILITY
I

25.4 Image.BrushType

NAME
Image.BrushType – set image type to use (V2.0)

FUNCTION
This attribute allows you to set the type of the Hollywood image object passed in the `Image.Brush` attribute. By default, `Image.Brush` expects a Hollywood brush. By setting `Image.BrushType`, however, you can make it use a different Hollywood image type.

The following image types are currently available:

`Brush` Use a Hollywood brush. This is the default type. You can use either raster or vector brushes. Vector brushes have the advantage that they can be scaled to any resolution without losses in quality. This is very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

`Icon` Use a Hollywood icon. This image type has the advantage that it can contain several subimages of different sizes. This makes it possible to provide images in different resolutions which can be very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

Note that you can globally change the default this attribute to Hollywood icons by setting the `Application.UseIcons` tag. See Section 9.13 [Application.UseIcons], page 65, for details.

TYPE
String (see above for possible values)

APPLICABILITY
I

26 Label class

26.1 Overview

Label class derives from Area class and creates labels for your widgets. Labels typically contain an underscore character to indicate the shortcut for accessing the widget that a label describes.

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

```
<hgroup>
    <label>_Name</label>
    <textentry/>
</hgroup>
```

The code above sets up a text entry widget that is labelled "Name". It also sets up a shortcut: When the user presses `ALT+N`, the text entry widget is automatically activated. See Section 3.13 [Keyboard shortcuts], page 21, 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 41.1 [Rectangle class], page 209, for details.

26.2 Label.Align

NAME
Label.Align – set label alignment

FUNCTION
Set the desired alignment for the label. This can be one of the following values:

`Left`	Left alignment.
`Right`	Right alignment. This is the default.
`Center`	Centered alignment.

TYPE
String (see above for possible values)

APPLICABILITY
I

26.3 Label.Text

NAME
Label.Text – set/get label text

FUNCTION
Set or get the label text. If the text contains an underscore, the character following the underscore will automatically be highlighted as the accelerator key. You can disable this behaviour by setting the `Area.NoAutoKey` attribute.

TYPE
String

APPLICABILITY
SG

27 Listview class

27.1 Overview

Listview class derives from Area class and shows a list container that can be filled with data items. RapaGUI's listview class is very powerful and supports multi-column lists, checkboxes, editable list items, data sorting via custom callbacks, and icons for the individual listview items.

When creating a listview in XML code, you always have to add at least one column to it. This is done by using 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>
```

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. For example, you can add checkboxes to your

columns and allow the editing of column items. See Section 28.1 [Listviewcolumn class], page 163, for details.

Note that RapaGUI might use up to three different kinds of widgets for this class: In case you are creating a listview that doesn't use any advanced functionality (such as multiple columns, icons, sorting) RapaGUI might create a more basic list widget for you because some operating systems offer several kinds of list-based widgets, e.g. on Windows there is a Listbox widget and a Listview widget. RapaGUI will use the Listbox widget in case your listview doesn't use any of the advanced features because listboxes are usually faster than listviews. If you don't want that, you can force RapaGUI to always give you a full-blown listview by setting the `Listview.ForceMode` attribute to the according tag. See Section 27.17 [Listview.ForceMode], page 146, for details.

27.2 Listview.AbortEditing

NAME

Listview.AbortEditing – get notified when user cancels editing (V1.1)

FUNCTION

When setting up a notification on this attribute, RapaGUI will run your event callback whenever the user cancels an editing operation on an item, for example by pressing the escape key or clicking outside the item edit widget.

Note that you have to set `Listviewcolumn.Editable` to `True` before you can use this attribute.

Your event handler will be called with the following extra arguments:

`Row:` Row index of the item which the user was editing when he cancelled the operation.

`Column:` Column index of the item which the user was editing when he cancelled the operation.

See Section 3.7 [Notifications], page 13, for details.

TYPE

Boolean

APPLICABILITY

N

27.3 Listview.Active

NAME

Listview.Active – set/get active list entry

FUNCTION

This attribute can be used to set or get the active list entry. This will always be between 0 and `Listview.Entries`-1 or the special value -1 in case there currently is no active entry.

If you set this attribute the listview will automatically scroll the position of the specified entry into view.

Besides an absolute index, you can also pass the following special values here:

`Off` Clear selection.

`Top` Select first entry.

`Bottom` Select last entry.

`Up` Select previous entry.

`Down` Select next entry.

`PageUp` Move list cursor one page up.

`PageDown` Move list cursor one page down.

You can also set up a notification on this attribute to get notified whenever the active entry is changed.

TYPE
Number or string (see above for possible values)

APPLICABILITY
ISGN

27.4 Listview.Alternate

NAME
Listview.Alternate – use alternating row colors

FUNCTION
Set this attribute to `True` to make the listview appear with alternating row colors.

Note that on AmigaOS and compatibles, this feature requires at least MUI 5.0.

TYPE
Boolean

APPLICABILITY
I

27.5 Listview.Clear

NAME
Listview.Clear – clear listview

SYNOPSIS
```
moai.DoMethod(id, "Clear")
```

FUNCTION
Remove all entries from listview.

INPUTS
`id` id of the listview object

27.6 Listview.ClickColumn

NAME
Listview.ClickColumn – learn about column clicks

FUNCTION
In multi-column listviews, this attribute records the number of the column where the user last clicked.

TYPE
Number

APPLICABILITY
GN

27.7 Listview.Columns

NAME
Listview.Columns – get number of columns (V2.0)

FUNCTION
Get number of listview columns.

TYPE
Number

APPLICABILITY
G

27.8 Listview.CompareItems

NAME
Listview.CompareItems – determine how entries should be sorted

FUNCTION
When setting up a notification on this attribute, RapaGUI will run your event callback whenever it 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.

Your event handler will be called with the following extra arguments:

`Entry1:` The first entry.

`Entry2:` The second entry.

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.

See Section 3.7 [Notifications], page 13, for details.

TYPE
Boolean

APPLICABILITY
N

27.9 Listview.DefClickColumn

NAME
Listview.DefClickColumn – set default column

PLATFORMS
AmigaOS and compatibles only

FUNCTION
When controlling the listview using the keyboard and pressing RETURN, the column number set here will be used as the default for `Listview.ClickColumn`.

TYPE
Number

APPLICABILITY
ISG

27.10 Listview.DoubleClick

NAME
Listview.DoubleClick – learn about double click on listview

FUNCTION
Set up a notification on this attribute to learn about double clicks on listview entries.

TYPE
Boolean

APPLICABILITY
N

27.11 Listview.DropFile

NAME
Listview.DropFile – learn about dropped files (V1.1)

FUNCTION
When setting up a notification on this attribute, RapaGUI will run your event callback whenever one or more files have been dropped onto the listview widget. The `TriggerValue` field of the message table will be set to a table that contains a list of all files that have been dropped onto the widget.

Additionally, the message table will contain the following two extra fields:

`X:` The x-position where the user has dropped the file(s). This will be relative to the widget's left corner.

`Y:` The y-position where the user has dropped the file(s). This will be relative to the widget's top corner.

See Section 3.7 [Notifications], page 13, for details.

Note that `Listview.DropFile` is only ever triggered if `Listview.DropTarget` has been set to `True` first.

TYPE
Boolean

APPLICABILITY
N

27.12 Listview.DropTarget

NAME
Listview.DropTarget – configure drop target settings (V1.1)

FUNCTION
Set this to `True` if files can be dropped on this listview widget. You can listen to the `Listview.DropFile` attribute to learn when the user drops one or more files on the listview widget.

TYPE
Boolean

APPLICABILITY
ISG

27.13 Listview.Edit

NAME
Listview.Edit – prompt user to edit an item

SYNOPSIS
```
moai.DoMethod(id, "Edit", row, column)
```

FUNCTION
This method can be used to programmatically initiate item editing. Normally, item editing is started by the user by slowly double-clicking an item. This method provides an alternative to this user mechanism.

This will only work if `Listviewcolumn.Editable` has been set to `True` for the respective listview column.

When the user has finished editing, the `Listview.ValueChange` attribute will be triggered.

Note that if you have installed a listener on the `Listview.StartEditing` attribute, then this callback will be asked for permission first before editing is actually started.

To learn about editing operations getting cancelled, you can listen to the `Listview.AbortEditing` attribute.

Also note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

INPUTS

`id`	id of the listview object
`row`	row index of the item to edit
`column`	column index of the item to edit

27.14 Listview.Entries

NAME

Listview.Entries – get listview entries

FUNCTION

Return the current number of entries in the listview.

TYPE

Number

APPLICABILITY

G

27.15 Listview.Exchange

NAME

Listview.Exchange – exchange two entries

SYNOPSIS

```
moai.DoMethod(id, "Exchange", pos1, pos2)
```

FUNCTION

Exchange two entries in a listview. The positions can be passed either as absolute values starting from 0 to `Listview.Entries`-1 or as one of the following special values:

`Top`	Use first entry.
`Active`	Use active entry.
`Bottom`	Use last entry.
`Next`	Use next entry. This is only valid in the second parameter.
`Previous`	Use previous entry. This is only valid in the second parameter.

This method can only be used with non-sortable listviews because the item order in sortable listviews is fixed and determined solely by the entries in the column that currently has the sort focus.

INPUTS

`id`	id of the listview object
`pos1`	number of the first entry
`pos2`	number of the second entry

27.16 Listview.First

NAME
Listview.First – get first visible entry

FUNCTION
Get the first visible listview entry.

TYPE
Number

APPLICABILITY
SG

27.17 Listview.ForceMode

NAME
Listview.ForceMode – override default listview mode

FUNCTION
RapaGUI can use up to three different widgets for Listview class depending on your settings. For example, in case a single-column list with no icons and headings is used, RapaGUI might use a different widget for reasons of efficiency in case the host OS provides such a widget. For example, on Windows RapaGUI will use the Listbox control instead of a fully featured Listview in those cases. If you don't want that, set this attribute to the desired widget and RapaGUI will try to use it.

The following modes are currently recognized:

`Normal`	Automatically selects the widget that fits best. This is the default.
`Listbox`	Use a Listbox widget. Listbox widgets only support a single column, no icons, no headings, no check boxes, no editable entries and no hidden columns.
`Listview`	Use a Listview widget. Listview widgets support everything instead of check boxes, editable entries, right and center alignment and hiding columns.
`Dataview`	Use a Dataview widget. Supports everything but currently uses a generic implementation on Windows.

Note that on AmigaOS and compatibles this attribute doesn't have any effect since RapaGUI always uses the same widget on those platforms.

TYPE
String (see above for possible values)

APPLICABILITY
I

27.18 Listview.GetColumnID

NAME
Listview.GetColumnID – get column id (V2.0)

SYNOPSIS
```
id$ = moai.DoMethod(id, "GetColumnID", pos)
```

FUNCTION
Returns the identifier of the column at the position specified by `pos`. `pos` can either be an absolute index starting from 0 for the first column or one of the following special values:

`First`	Use first column.
`Last`	Use last column.

INPUTS

`id`	id of the listview object
`pos`	absolute index of column or special value (see above)

RESULTS

`id$`	id of the column at the specified index

27.19 Listview.GetDisabled

NAME
Listview.GetDisabled – get checkbox disabled state

SYNOPSIS
```
state = moai.DoMethod(id, "GetDisabled", row, column)
```

FUNCTION
Returns the disabled state of the checkbox in the specified row and column. This is either `True` or `False`.

INPUTS

`id`	id of the listview object
`row`	row index of the checkbox
`column`	column index of the checkbox

RESULTS

`state`	`True` if the checkbox is disabled, `False` otherwise

27.20 Listview.GetEntry

NAME

Listview.GetEntry – get listview entry

SYNOPSIS

```
column1$, ... = moai.DoMethod(id, "GetEntry", pos[, icons])
```

FUNCTION

Get an entry from a listview. You can pass either an absolute index in `pos` or the special value `Active` to get the active entry. `Listview.GetEntry` will then 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.

Starting with RapaGUI 2.0, `Listview.GetEntry` accepts an optional `icons` argument now. If this is set to `True`, `Listview.GetEntry` will also return the identifier of the brush/icon used for a listview item or -1 if no brush/icon has been set for this item. Note that the identifier is only returned for columns that have the `Listviewcolumn.Icon` attribute set to `True`.

INPUTS

`id`	id of the listview object
`pos`	index of listview row or "Active"
`icons`	optional: `True` if information about icons should be returned as well (V2.0)

RESULTS

`column1$`	entry data of first column
`...`	further data if listview has multiple columns

27.21 Listview.GetSelection

NAME

Listview.GetSelection – get selected entries

SYNOPSIS

```
t = moai.DoMethod(id, "GetSelection")
```

FUNCTION

Returns a table containing all selected entries of a multi-select listview. Note that this should only be used with multi-select listviews. For single-select listviews, you can use `Listview.Active` to get the selected entry.

INPUTS

`id`	id of the listview object

RESULTS

`t`	table containing selected entries

27.22 Listview.GetState

NAME
Listview.GetState – get checkbox toggle state

SYNOPSIS
```
state = moai.DoMethod(id, "GetState", row, column)
```

FUNCTION
Returns the toggle state of the checkbox in the specified row and column. This is either `True` if the checkbox is selected or `False` otherwise.

INPUTS

`id`	id of the listview object
`row`	row index of the checkbox
`column`	column index of the checkbox

RESULTS

`state`	`True` if the checkbox is selected, `False` otherwise

27.23 Listview.HRules

NAME
Listview.HRules – draw horizontal rules between rows

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set this to `True` to enable horizontal rules between rows for the listview.

TYPE
Boolean

APPLICABILITY
I

27.24 Listview.Insert

NAME
Listview.Insert – insert new entry

SYNOPSIS
```
pos = moai.DoMethod(id, "Insert", pos, [icon1,] column1$, ...)
```

FUNCTION
Insert one new entry into the listview. If the listview has multiple columns, you need to pass individual entry data for all the columns in the listview.

The entry data consists of a text string and, if the column has the `Listviewcolumn.Icon` attribute set, an icon for each column. The icon has to be passed before the text string

and it has to be an identifier of a Hollywood brush/icon which should be used as the icon for the entry. If `Listviewcolumn.Icon` isn't set, then you must omit the icon parameter and only pass text data for the listview entry. If you've set `Listviewcolumn.Icon` to `True` and you don't want to show an icon in this particular row and column, you can also pass the special value -1. In that case, RapaGUI won't show an icon even though `Listviewcolumn.Icon` has been set to `True`. Please note that auto-generated IDs cannot be used. Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

In case a column is showing a checkbox, you have to pass `"On"`, `"True"`, or `"1"` to select the checkbox and any other text to deselect the checkbox.

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 before the active entry. If there is no active entry, the entry will be inserted at the very top of the list.

`Bottom` Insert as last entry.

If `pos` is bigger or equal to the number of entries in the listview, the entry will be inserted as the last entry.

If the listview has sortable columns, `pos` will be ignored and the entry will automatically be sorted into the listview, depending on which column currently has the sort focus. See Section 27.40 [Listview.SortColumn], page 158, for details.

`Listview.Insert` returns the position of the newly inserted entry. This is especially useful when the listview has sortable columns because in that case, you can't easily compute where the entry will end up in the listview.

Note that on AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

INPUTS

`id` id of the listview object

`pos` insert position as absolute number or special value (see above)

`icon1` optional: icon for the first column; this must only be passed if the column has the icon attribute set; note that this must be a numeric identifier and auto-generated IDs are not valid in this case

`column1$` entry to insert into first column

`...` more entries if listview has multiple columns

RESULTS

`pos` position of the newly inserted entry (starting from 0)

27.25 Listview.InsertColumn

NAME

Listview.InsertColumn – insert new column (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "InsertColumn", pos, cid$[, t])
```

FUNCTION

Insert a new column into the listview. The column will be inserted into the position specified by `pos` and will be given the identifier specified by `cid$`. The new column will be added in front of the column specified by `pos`. This can be an absolute index position starting at 0 for the first column or one of the following special values:

`First` Insert as first column.

`Last` Insert as last column.

The optional argument `t` can be set to a table containing one or more of the following options:

`Align` Desired column alignment. See Section 28.2 [Listviewcolumn.Align], page 163, for details.

`Checkbox` Flag indicating if column should use checkboxes. See Section 28.3 [Listviewcolumn.Checkbox], page 163, for details.

`Editable` Flag indicating if column should be editable. See Section 28.4 [Listviewcolumn.Editable], page 164, for details.

`Hide` Flag indicating if column should be hidden. See Section 28.5 [Listviewcolumn.Hide], page 164, for details.

`Icon` Flag indicating whether column should use icons. See Section 28.6 [Listviewcolumn.Icon], page 165, for details.

`IconType` Icon type to use. See Section 28.8 [Listviewcolumn.IconType], page 166, for details.

`IconScale`
Toggle icon scaling. See Section 28.7 [Listviewcolumn.IconScale], page 165, for details.

`Sortable` Flag indicating if column should be sortable. See Section 28.9 [Listviewcolumn.Sortable], page 167, for details.

`SortOrder`
Desired column sort order. See Section 28.10 [Listviewcolumn.SortOrder], page 167, for details.

`Title` Desired column title. See Section 28.11 [Listviewcolumn.Title], page 167, for details.

`Width` Desired column width. See Section 28.12 [Listviewcolumn.Width], page 168, for details.

Note that `Listview.InsertColumn` is only supported if the listview is in `Listview` or `Dataview` mode. See Section 27.17 [Listview.ForceMode], page 146, for details.

INPUTS

`id`	id of the listview object
`pos`	insert position as absolute number or special value (see above)
`cid$`	identifier for new column
`t`	optional: table containing further parameters

27.26 Listview.ItemStyle

NAME
Listview.ItemStyle – set listview item style (V2.0)

PLATFORMS
Android

FUNCTION
On Android, you can specify a style that should be used for the listview items. This can be one of the following special values:

`Default`	The default style.
`Normal`	Normal list item style. This corresponds to the style `android.R.layout.simple_list_item_1`.
`Big`	Use big list item. This corresponds to the style `android.R.layout.simple_expandable_list_item_1`.

TYPE
String (see above for possible values)

APPLICABILITY
I

27.27 Listview.Jump

NAME
Listview.Jump – scroll to an entry

SYNOPSIS
```
moai.DoMethod(id, "Jump", pos)
```

FUNCTION
Scroll the specified entry into the visible part of the listview. `pos` can be an absolute index or one of the following special values:

`Top`	Scroll top entry into view.
`Active`	Scroll active entry into view.

`Bottom` Scroll last entry into view.

INPUTS

`id` id of the listview object

`pos` number of the entry that should be made visible or special value (see above)

27.28 Listview.LongClick

NAME
Listview.LongClick – learn about long click on listview (V2.0)

PLATFORMS
Android

FUNCTION
Set up a notification on this attribute to learn about long clicks on listview entries. Long clicks are currently only triggered on Android when the user taps a listview item for a longer time.

TYPE
Boolean

APPLICABILITY
N

27.29 Listview.Move

NAME
Listview.Move – move entry to new position

SYNOPSIS
```
moai.DoMethod(id, "Move", from, to)
```

FUNCTION
Move the specified entry to a new position. 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 first entry.

`Active` Use active entry.

`Bottom` Use last 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.

This method can only be used with non-sortable listviews because the item order in sortable listviews is fixed and determined solely by the entries in the column that currently has the sort focus.

INPUTS

`id`	id of the listview object
`from`	number of the first entry
`to`	number of the second entry

27.30 Listview.MultiSelect

NAME
Listview.MultiSelect – enable multi select mode

FUNCTION
Set this attribute to `True` to allow the selection of multiple entries in this listview. By default, only a single entry can be selected at a time.

TYPE
Boolean

APPLICABILITY
I

27.31 Listview.Quiet

NAME
Listview.Quiet – disable listview refresh

FUNCTION
Adding or removing many entries at once is usually quite expensive because the widget is refreshed after every add and remove operation. Setting `Listview.Quiet` to `True` will temporarily turn off listview refresh until you set the attribute to `False` again. This can speed up insertion and remove operations significantly in case lots of entries are affected.

TYPE
Boolean

APPLICABILITY
S

27.32 Listview.Remove

NAME
Listview.Remove – remove entry from listview

SYNOPSIS

```
moai.DoMethod(id, "Remove", pos)
```

FUNCTION

Remove an entry from a listview. 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.

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)

27.33 Listview.RemoveColumn

NAME

Listview.RemoveColumn – remove column (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "RemoveColumn", pos)
```

FUNCTION

Remove the column at index `pos` from the listview. `pos` can be an absolute index position starting from 0 for the first column or one of the following special values:

`First` Remove first column.

`Last` Remove last column.

Note that `Listview.RemoveColumn` is only supported if the listview is in `Listview` or `Dataview` mode. See Section 27.17 [Listview.ForceMode], page 146, for details.

INPUTS

`id` id of the listview object

`pos` remove position as absolute index or special value (see above)

27.34 Listview.Rename

NAME

Listview.Rename – rename an entry

SYNOPSIS

```
moai.DoMethod(id, "Rename", pos, [icon1,] column1$, ...)
```

FUNCTION

Rename the listview entry at the specified position. If the listview has multiple columns, you need to pass a new name for every column. 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.

For all columns that have the `Listviewcolumn.Icon` attribute set, you also need to pass an icon before the actual text. The icon has to be an identifier of a Hollywood brush/icon which should be used as the icon for the column. If `Listviewcolumn.Icon` isn't set, then you must omit the icon parameter and only pass text data for the listview entry. If you've set `Listviewcolumn.Icon` to `True` and you don't want to show an icon in this particular row and column, you can also pass the special value -1. In that case, RapaGUI won't show an icon even though `Listviewcolumn.Icon` has been set to `True`. Please note that auto-generated IDs cannot be used. Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

In case a column is showing a checkbox, you have to pass `"On"`, `"True"`, or `"1"` to select the checkbox and any other text to deselect the checkbox.

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.

Note that on AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

INPUTS

`id` id of the listview object

`pos` entry position as absolute number or special value (see above)

`icon1` optional: icon for the first column; this must only be passed if the column has the icon attribute set; note that this must be a numeric identifier and auto-generated IDs are not valid in this case

`column1$` new text for entry in the first column

`...` more entries if listview has multiple columns

27.35 Listview.RowHeight

NAME

Listview.RowHeight – set row height (V2.0)

FUNCTION

Set listview row height. Normally, this is not necessary because the row height is calculated automatically. However, it might be useful to set a specific row height when using very large icons that would otherwise be cut off.

TYPE

Number

APPLICABILITY

I

27.36 Listview.Select

NAME

Listview.Select – (de)select listview entry

SYNOPSIS

```
state = moai.DoMethod(id, "Select", pos, seltype)
```

FUNCTION

Select or deselect a listview entry or query the selection state of an entry.

`Pos` can be either the number of the entry or one of the following special values:

`Active`	Use the active entry.
`All`	Use all entries.

`Seltype` can be one of the following:

`Off`	Deselect entry.
`On`	Select entry.
`Toggle`	Toggle entry.
`Ask`	Query selection state of specified entry. If this is set, Listview.Select returns the selection state of the specified entry (either 1 or 0).

INPUTS

`id`	id of the listview object
`pos`	entry index or special value (see above)
`seltype`	selection type (see above)

RESULTS

`state`	selection state of entry; this is only valid when using "Ask" for `seltype` (see above)

27.37 Listview.SetDisabled

NAME

Listview.SetDisabled – set checkbox disabled state

SYNOPSIS

```
moai.DoMethod(id, "SetDisabled", row, column, state)
```

FUNCTION

Sets the disabled state of the checkbox in the specified row and column. Pass `True` to disable the checkbox or `False` to enable it.

INPUTS

`id`	id of the listview object
`row`	row index of the checkbox
`column`	column index of the checkbox
`state`	`True` if the checkbox should be disabled, `False` otherwise

27.38 Listview.SetState

NAME
Listview.SetState – set checkbox toggle state

SYNOPSIS
`moai.DoMethod(id, "SetState", row, column, state)`

FUNCTION
Sets the toggle state of the checkbox in the specified row and column. Pass `True` to select the checkbox or `False` to unselect it.

INPUTS

`id`	id of the listview object
`row`	row index of the checkbox
`column`	column index of the checkbox
`state`	`True` if the checkbox should be selected, `False` otherwise

27.39 Listview.Sort

NAME
Listview.Sort – sort the entries

SYNOPSIS
`moai.DoMethod(id, "Sort")`

FUNCTION
Sort all entries of the listview. It's normally not necessary to call this function because the listview will automatically sort all items as soon as there is a column that is sortable.

INPUTS

`id`	id of the listview object

27.40 Listview.SortColumn

NAME
Listview.SortColumn – set/get sort column (V2.0)

FUNCTION
Set or get the column that should be used for sorting the listview entries. This must be set to the index of the column that should get the sort focus. Column indices start at 0 for the first column. Note that you can only set columns as the sort columns that have `Listviewcolumn.Sortable` set to `True`. The sort order can be configured by setting the `Listviewcolumn.SortOrder` attribute.

Note that the user can also set the sort column by clicking on its header. If you'd like to be notified about such events, you can install a listener on this attribute.

TYPE
Number

APPLICABILITY
ISGN

27.41 Listview.StartEditing

NAME
Listview.StartEditing – get notified about item editing

FUNCTION
When setting up a notification on this attribute, RapaGUI will run your event callback whenever the user wants to edit a listview item by slowly double-clicking on it or when your script runs the `Listview.Edit` method. Your callback can then permit or forbid the user's edit request. To forbid the request, your callback has to return `False`. To permit the request, it has to return `True`.

Note that you have to set `Listviewcolumn.Editable` to `True` before you can use this attribute.

Your event handler will be called with the following extra arguments:

`Row:` Row index of the item which the user wants to edit.

`Column:` Column index of the item which the user wants to edit.

See Section 3.7 [Notifications], page 13, for details.

TYPE
Boolean

APPLICABILITY
N

27.42 Listview.SystemTheme

NAME
Listview.SystemTheme – make listview use the system theme (V2.0)

PLATFORMS
Windows

FUNCTION
Set this to `True` to make the listview use the system's theme. This is only supported if `Listview.ForceMode` is set to `Listview` or if RapaGUI implicitly sets `Listview.ForceMode` to `Listview`.

TYPE
Boolean

APPLICABILITY
I

27.43 Listview.TitleClick

NAME
Listview.TitleClick – learn about title clicks

FUNCTION
This attribute is set to the column index number whenever the user clicks on a column title button.

On AmigaOS and compatibles this attribute requires at least MUI 4.0.

TYPE
Number

APPLICABILITY
N

27.44 Listview.ValueChange

NAME
Listview.ValueChange – get notified when listview item value changes

FUNCTION
When setting up a notification on this attribute, RapaGUI will run your event callback whenever a listview item's value has changed because the user has toggled the checkbox or has edited the item. Note that `Listview.ValueChange` will not trigger if the item's value was changed using the `Listview.Rename` method.

For items in checkbox columns, `TriggerValue` will be set to either `True` or `False`, reflecting the new checkbox state. For items in text columns, `TriggerValue` will contain the new item text.

Additionally, your event handler will be called with the following extra arguments:

`Row:` Row index of the item whose value has changed.

`Column:` Column index of the item whose value has changed.

See Section 3.7 [Notifications], page 13, for details.

TYPE
Boolean or string (depending on column type)

APPLICABILITY
N

27.45 Listview.Visible

NAME
Listview.Visible – get number of visible entries

FUNCTION
Get the number of entries in the listview that are currently visible.

TYPE
Number

APPLICABILITY
G

27.46 Listview.VRules

NAME
Listview.VRules – draw vertical rules between columns

FUNCTION
Set this to `True` to enable vertical rules between columns for the listview.

TYPE
Boolean

APPLICABILITY
I

28 Listviewcolumn class

28.1 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. Its XML tag is `<column>`. See Section 27.1 [Listview class], page 139, for details.

Note that you cannot create instances of this class using `moai.CreateObject()`. Listview column numbers are currently static, i.e. you cannot add or remove columns at runtime.

28.2 Listviewcolumn.Align

NAME
Listviewcolumn.Align – set/get column alignment

FUNCTION
Set or get the column alignment. This can be one of the following values:

`Left` Left alignment. This is the default.

`Right` Right alignment.

`Center` Centered alignment.

On non-AmigaOS systems this attribute is only supported for the dataview backend. When creating a listview and `Listviewcolumn.Align` is set to a value other than `Left`, RapaGUI will automatically switch to the dataview backend. If you don't specify this attribute at creation time but want to set it later using `moai.Set()`, you have to explicitly request a dataview widget by setting the `Listview.ForceMode` attribute.

TYPE
String (see above for possible values)

APPLICABILITY
ISG

28.3 Listviewcolumn.Checkbox

NAME
Listviewcolumn.Checkbox – put column in checkbox mode

FUNCTION
Set this to `True` to mark this column as a checkbox column. Checkbox columns show checkboxes instead of text. Whenever an item's text in a checkbox column is set to `"On"`, `"True"`, or `"1"`, the checkbox will be selected. All other item texts will lead to an unselected checkbox.

You can modify the states of the checkboxes either by using the `Listview.Rename` method or by using the dedicated `Listview.SetState` method. Similarly, getting the

state of a checkbox is possible via the `Listview.GetEntry` or `Listview.GetState` methods.

To get notified whenever the user toggles a checkbox state, you have to listen to the `Listview.ValueChange` attribute.

Also note that `Listviewcolumn.Checkbox` and `Listviewcolumn.Editable` and `Listviewcolumn.Icon` are mutually exclusive. You cannot create checkbox columns that are editable or show icons.

TYPE
Boolean

APPLICABILITY
IG

28.4 Listviewcolumn.Editable

NAME
Listviewcolumn.Editable – allow editing of column items

FUNCTION
Set this to `True` to allow user editing of the items in this column. The user will then be able to edit all items in this column by executing a slow double click, i.e. slowly pressing left mouse button two times in a row.

If you would only like to allow editing of some items in the column, you need to set this attribute to `True` and you need to listen to the `Listview.StartEditing` attribute. `Listview.StartEditing` will then be triggered whenever the user attempts to edit an item and your callback can return `False` to forbid editing of certain items. See Section 27.41 [Listview.StartEditing], page 159, for details.

To get notified whenever the value of a listview item changes because the user has edited it, you have to listen to the `Listview.ValueChange` attribute.

To manually start editing of a listview item, call the `Listview.Edit` method.

Also, `Listviewcolumn.Checkbox` and `Listviewcolumn.Editable` are mutually exclusive. You cannot create editable checkbox columns.

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

TYPE
Boolean

APPLICABILITY
IG

28.5 Listviewcolumn.Hide

NAME
Listviewcolumn.Hide – show/hide listview column

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.

On non-AmigaOS systems this attribute is only supported for the dataview backend. When creating a listview and `Listviewcolumn.Hide` is set to `True`, RapaGUI will automatically switch to the dataview backend. If you don't specify this attribute at creation time but want to set it later using `moai.Set()`, you have to explicitly request a dataview widget by setting the `Listview.ForceMode` attribute.

TYPE
Boolean

APPLICABILITY
ISG

28.6 Listviewcolumn.Icon

NAME
Listviewcolumn.Icon – enable icons for this column

FUNCTION
Set this to `True` if listview entries in this column use icons. In that case, you have to pass Hollywood brushes/icons to use as icons to the `Listview.Insert` method.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

On AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

TYPE
Boolean

APPLICABILITY
IG

28.7 Listviewcolumn.IconScale

NAME
Listviewcolumn.IconScale – configure automatic image scaling (V2.0)

FUNCTION
If `Listviewcolumn.Icon` has been set to a raster brush and RapaGUI is running on a high-DPI display, RapaGUI will automatically scale the brush's raster graphics to fit to the current monitor's DPI setting. If you don't want that, set this tag to `False`.

Alternatively, you can also globally disable automatic image scaling by setting the `ScaleGUI` tag to `False` when calling `@REQUIRE` on RapaGUI. See Section 3.4 [Initializing RapaGUI], page 11, for details.

Please also read the chapter about high-DPI support in RapaGUI to learn more about supporting high-DPI displays. See Section 3.12 [High-DPI support], page 20, for details.

TYPE
Boolean

APPLICABILITY
I

28.8 Listviewcolumn.IconType

NAME
Listviewcolumn.IconType – set icon type to use (V2.0)

FUNCTION
This attribute allows you to set the type of the Hollywood image object passed in the `Listviewcolumn.Icon` attribute. By default, `Listviewcolumn.Icon` expects a Hollywood brush. By setting `Listviewcolumn.IconType`, however, you can make it use a different Hollywood image type.

The following image types are currently available:

`Brush` Use a Hollywood brush. This is the default type. You can use either raster or vector brushes. Vector brushes have the advantage that they can be scaled to any resolution without losses in quality. This is very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

`Icon` Use a Hollywood icon. This image type has the advantage that it can contain several subimages of different sizes. This makes it possible to provide images in different resolutions which can be very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

Note that you can globally change the default of all `IconType` attributes to Hollywood icons by setting the `Application.UseIcons` tag. See Section 9.13 [Application.UseIcons], page 65, for details.

TYPE
String (see above for possible values)

APPLICABILITY
I

28.9 Listviewcolumn.Sortable

NAME
Listviewcolumn.Sortable – make column sortable

FUNCTION
Set this to `True` to mark the column as sortable. You can then set this column as the current sort column by setting the `Listview.SortColumn` attribute.

TYPE
Boolean

APPLICABILITY
ISG

28.10 Listviewcolumn.SortOrder

NAME
Listviewcolumn.SortOrder – set/get column sort order (V2.0)

FUNCTION
Set or get the desired sort order for the column. This can be one of the following special values:

`Ascending`
: Ascending order. This is the default.

`Descending`
: Descending order.

The user can also change the sort order by clicking on a listview column's header. If you set up a notification on this attribute, you will be notified about such events.

TYPE
String (see above for possible values)

APPLICABILITY
ISGN

28.11 Listviewcolumn.Title

NAME
Listviewcolumn.Title – set/get column title

FUNCTION
Set or get the title for the column. The title is always shown at the top of the listview and doesn't go away when the listview is scrolled.

TYPE
String

APPLICABILITY
ISG

28.12 Listviewcolumn.Width

NAME
Listviewcolumn.Width – set/get column width

FUNCTION
Set or get the column width in device-independent pixels. This defaults to -1 which means that the column should be made as large as its largest entry.

TYPE
Number

APPLICABILITY
ISG

29 Listviewitem class

29.1 Overview

Listviewitem class can be used when creating listviews to add listview entries already at object creation time. See Section 27.1 [Listview class], page 139, for details.

Listviewitem class entries must always be embedded inside a `<column>` declaration. Its XML tag is `<item>`. See Section 28.1 [Listviewcolumn class], page 163, for details.

Note that you cannot create instances of this class using `moai.CreateObject()`. Instead, you have to use `Listview.Insert` to create new items at runtime. Also, you cannot change any attributes of listview items using this class. If you want to change the text or the icon of a listview entry, you need to use `Listview.Rename` instead.

29.2 Listviewitem.Icon

NAME

Listviewitem.Icon – set image for listview entry

FUNCTION

Set this attribute to the identifier of a Hollywood brush or icon to add an image to your listview entry. Whether this attribute expects a Hollywood brush or icon, depends on what you specify in the `Listviewcolumn.IconType` attribute. By default, `Listviewitem.Icon` expects a Hollywood brush. You also need to set `Listviewcolumn.Icon` to `True` if you want to use icons in a listview column.

To change the icon of a listview entry later, you can use the `Listview.Rename` method.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

On AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

TYPE

Number

APPLICABILITY

I

30 Menu class

30.1 Overview

Menu class can be used to create a single pull-down menu which should then be filled with items derived from Menuitem class class. Menus are usually embedded inside a top-level `<menubar>` object. See Section 31.1 [Menubar class], page 175, for details. Alternatively, they can also be embedded as submenus of other menu objects.

It is also possible to add a menu object as a context menu to one of your window's widgets using the `Area.ContextMenu` attribute. The context menu will then appear whenever the user presses the right mouse button over the widget that hosts the context menu. Here is an example of how to add a menu object as a context menu to a text editor widget:

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

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

Note that when using Menu class to create context menus, the `Menu.Title` attribute is only used on AmigaOS and compatibles. Context menus on Windows, Linux, and macOS don't show a title.

30.2 Menu.Append

NAME

Menu.Append – add detached object as last menu child

SYNOPSIS

```
moai.DoMethod(id, "Append", obj)
```

FUNCTION

This method can be used to add the detached object specified by `obj` to the menu object specified by `id`. The detached object will be added as the menu'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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Menu.Remove` method.

INPUTS

`id` id of the menu object

`obj`	id of the object to attach

30.3 Menu.Disabled

NAME
Menu.Disabled – set/get disabled state of menu

FUNCTION
Enable or disable the complete menu.

TYPE
Boolean

APPLICABILITY
ISG

30.4 Menu.Insert

NAME
Menu.Insert – insert detached object after specified child

SYNOPSIS
```
moai.DoMethod(id, "Insert", obj, pred)
```

FUNCTION
This method can be used to insert the detached object specified by `obj` to the menu 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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Menu.Remove` method.

INPUTS

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

30.5 Menu.NoAutoKey

NAME
Menu.NoAutoKey – disable automatic shortcut generation

FUNCTION
Set this attribute to disable automatic shortcut generation. By default, the character following an underscore character in a menu title will be treated as an accelerator key that

can be used to access the menu entry using the keyboard. If you set this attribute, underscore characters will never be treated as shortcut markers and will be shown normally in menu entries.

See Section 3.13 [Keyboard shortcuts], page 21, for details.

TYPE
Boolean

APPLICABILITY
I

30.6 Menu.Prepend

NAME
Menu.Prepend – add detached object as first menu child

SYNOPSIS
```
moai.DoMethod(id, "Prepend", obj)
```

FUNCTION
This method can be used to add the detached object specified by `obj` to the menu object specified by `id`. The detached object will be added as the menu'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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Menu.Remove` method.

INPUTS

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

30.7 Menu.Remove

NAME
Menu.Remove – detach object from menu

SYNOPSIS
```
moai.DoMethod(id, "Remove", obj)
```

FUNCTION
This method can be used to detach the specified object from the specified menu. 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 menu using a function like `Menu.Insert` or you could free it using `moai.FreeObject()`.

INPUTS

`id`	id of the menu object

`obj` id of the object to remove

30.8 Menu.Title

NAME
Menu.Title – set/get menu title

FUNCTION
Set or get the title of the menu.

If the title contains an underscore, RapaGUI will automatically set up the character following this underscore as a keyboard shortcut. If you don't want this behaviour, set `Menu.NoAutoKey` to `True`.

TYPE
String

APPLICABILITY
ISG

30.9 Menu.Type

NAME
Menu.Type – set menu type (V2.0)

FUNCTION
This attribute can be used to set the type for this menu. The following types are currently possible:

`Help` Designate item as the help menu. Only supported on macOS.

Note that when setting this attribute to one of the types only supported on macOS, RapaGUI will move the menu to its appropriate position according to macOS' styleguide, e.g. the "Help" menu always appears at the far right of the menubar. On non-macOS systems, those special types don't have any effect and RapaGUI will just ignore them.

TYPE
String (see above for possible values)

APPLICABILITY
I

31 Menubar class

31.1 Overview

Menubar class is used to manage top-level menus that can be attached to windows via the `Window.Menubar` attribute. A menubar contains a number of children which are objects of Menu class, each of them describing exactly one menu.

In an XML file a menu tree is defined using the `<menubar>`, `<menu>` and `<item>` tags. Here is an example definition of a simple menubar:

```
<menubar id="mymenubar">
    <menu title="_File">
        <item>_New...</item>
        <item>_Open...</item>
        <item/>
        <item>_Save</item>
        <item>S_ave as...</item>
        <item/>
        <item>_Quit</item>
    </menu>
    <menu title="Edit">
        <item shortcut="Ctrl+X">_Cut</item>
        <item shortcut="Ctrl+C">C_opy</item>
        <item shortcut="Ctrl+V">_Paste</item>
    </menu>
    <menu title="?">
        <item>Se_ttings...</item>
        <item/>
        <item>A_bout...</item>
        <item>About _RapaGUI...</item>
    </menu>
</menubar>
```

Note the use of the underscore character in the XML code above: You can use this character to automatically designate the next character as a shortcut key for the menu item. This is very useful because many people like to use the keyboard instead of the mouse, especially when it comes to repeating the same actions many times. Thus, it is always a good idea to set up keyboard shortcuts. See Section 3.13 [Keyboard shortcuts], page 21, for details.

If you need more complex keyboard shortcuts, e.g. something like `Ctrl+V` for paste, you can use the attribute `Menuitem.Shortcut` to set up such a shortcut. Note that on some platforms (e.g. Windows), both types of shortcuts can be specified at the same time: Shortcuts specified by using the underscore character and shortcuts declared via the `Menuitem.Shortcut`. If a certain platform supports only one type of shortcut, the one specified in `Menuitem.Shortcut` will take precedence over the one specified using the underscore character.

Also note the empty `<item/>` declarations: These will insert a separator bar into 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 menubar to one of your windows by using the `Window.Menubar` attribute as follows:

```
<window menubar="mymenubar">
...
</window>
```

It is very important to note that you have to declare your menubars in the `<application>` scope because menubars are global objects and are only attached to windows or widgets later on. That is why it is not allowed to declare menubars inside a `<window>` XML scope.

31.2 Menubar.Append

NAME

Menubar.Append – add detached object as last menubar child

SYNOPSIS

```
moai.DoMethod(id, "Append", obj)
```

FUNCTION

This method can be used to add the detached object specified by `obj` to the menubar object specified by `id`. The detached object will be added as the menubar'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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Menubar.Remove` method.

INPUTS

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

31.3 Menubar.Insert

NAME

Menubar.Insert – insert detached object after specified child

SYNOPSIS

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

FUNCTION

This method can be used to insert the detached object specified by `obj` to the menubar 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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Menubar.Remove` method.

INPUTS

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

31.4 Menubar.Prepend

NAME

Menubar.Prepend – add detached object as first menubar child

SYNOPSIS

```
moai.DoMethod(id, "Prepend", obj)
```

FUNCTION

This method can be used to add the detached object specified by `obj` to the menubar object specified by `id`. The detached object will be added as the menubar'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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Menubar.Remove` method.

INPUTS

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

31.5 Menubar.Remove

NAME

Menubar.Remove – detach object from menubar

SYNOPSIS

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

FUNCTION

This method can be used to detach the specified object from the specified menubar. 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 menubar using a function like `Menubar.Insert` or you could free it using `moai.FreeObject()`.

INPUTS

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

32 Menuitem class

32.1 Overview

Menuitem class can be used to create a single menu item. As such menuitems must always be embedded inside a `<menu>` tag which in turn is usually embedded inside a `<menubar>` tag. See Section 31.1 [Menubar class], page 175, for details.

Please note that the XML tag for menuitem class is just `<item>` and not `<menuitem>` when creating menu items that are embedded inside a `<menu>` tree. When creating isolated menu items that aren't embedded inside a `<menu>` tree, you have to use `<menuitem>`, though because otherwise RapaGUI wouldn't be able to know which class you are referring to since `<item>` is used by many different classes.

If you do not specify a title for the menuitem, a separator item will be created. This is useful to group certain menuitems together which results in a more readable menu layout.

See Section 31.1 [Menubar class], page 175, for an example.

32.2 Menuitem.Disabled

NAME
Menuitem.Disabled – set/get disabled state of menu item

FUNCTION
Set or get the menu item's disabled state.

TYPE
Boolean

APPLICABILITY
ISG

32.3 Menuitem.Help

NAME
Menuitem.Help – set/get menu item help text

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set or get the menu item's help text. This text is automatically shown in the status bar of the window that hosts the menu whenever the mouse hovers over the menu item. This is quite convenient for the user because it explains the function of the individual menu items in a somewhat greater detail. See Section 46.1 [Statusbar class], page 227, for details.

TYPE
String

APPLICABILITY
ISG

32.4 Menuitem.NoAutoKey

NAME
Menuitem.NoAutoKey – disable automatic shortcut generation

FUNCTION
Set this attribute to disable automatic shortcut generation. By default, the character following an underscore character in a menu item title will be treated as an accelerator key that can be used to access the menu item using the keyboard. If you set this attribute, underscore characters will never be treated as shortcut markers and will be shown normally in menu items.

See Section 3.13 [Keyboard shortcuts], page 21, for details.

TYPE
Boolean

APPLICABILITY
I

32.5 Menuitem.Selected

NAME
Menuitem.Selected – learn if an item is selected

FUNCTION
This attribute is triggered when the user selects a menu item. RapaGUI automatically listens to this attribute for all menu items so you do not need to explicitly request a notification using the MOAI.Notify attribute.

For menu items that can be toggled or are part of a radio group, querying this attribute returns the current toggle or radio state of the menu item.

TYPE
Boolean

APPLICABILITY
ISGN

32.6 Menuitem.Shortcut

NAME
Menuitem.Shortcut – set custom shortcut for menu item

FUNCTION
Set a custom shortcut for a menu item. Normally, menu item shortcuts are defined by just using the underscore character in the item title to designate a keyboard shortcut.

See Section 3.13 [Keyboard shortcuts], page 21, for details. There are however certain cases where this is not sufficient. First of all, by using the underscore character to declare a keyboard shortcut you are limited to characters which are part of the menu item's title. In many times, though, you will want to declare a shortcut that is not part of the item's title, e.g. the typical shortcut for `"Paste"` is `Ctrl+V` although there is no `"V"` in the word `"Paste"`. This is where `Menuitem.Shortcut` can be used to set up arbitrary shortcuts for your menu items.

Additionally, this attribute is also useful for defining more complex shortcuts, e.g. `Alt+F5` or `Ctrl+Shift+X`, etc. Defining such complex shortcuts isn't possible through the standard underscore route either, so you have to use `Menuitem.Shortcut` for it.

The string you specify here must be combination of modifiers and keys separated by `+` or `-` characters. The following modifiers and keys are currently recognized:

`Ctrl` This specifies the control key on Windows and Linux and the command key on AmigaOS and macOS. The reason why this is mapped to the command key on AmigaOS and macOS is to make it easier for you to write portable code. On Windows and Linux the control key is the standard modifier key whereas AmigaOS and macOS use the command key as the standard modifier key. Thus, by specifying this modifier you will always get the system's default modifier key. If you need to use the real control key on AmigaOS and macOS, use the modifier key `RawCtrl` (see below). If RapaGUI didn't treat `Ctrl` as a special token, you'd always have to write separate code for Windows and Linux vs. AmigaOS and macOS, i.e. you'd have to specify `Ctrl+V` for paste on Windows and Linux and `Cmd+V` for paste on AmigaOS and macOS. Since this generates unnecessary overhead, `Ctrl` is treated as a special key which is mapped to the system's default menu shortcut modifier key.

`Alt` Use the alt key as a modifier key.

`Shift` Use the shift key as a modifier key.

`RawCtrl` On AmigaOS and macOS this modifier key allows you to listen to the real control key. If you use `Ctrl` instead, RapaGUI will listen to the command key on AmigaOS and macOS (see above for details). On Windows and Linux `RawCtrl` is identical to `Ctrl`.

`Up` Cursor up

`Down` Cursor down

`Right` Cursor right

`Left` Cursor left

`Help` Help key

`Del` Delete key

`Backspace`
Backspace key

`Tab` Tab key

`Return` Return key

`Enter`	Enter key
`Esc`	Escape
`Space`	Space key
`F1 - F16`	Function keys
`Insert`	Insert key
`Home`	Home key
`End`	End key
`PageUp`	Page up key
`PageDown`	Page down key
`Print`	Print key
`Pause`	Pause key

Furthermore, you can also add English alphabet characters from A to Z as well as the numbers 0 to 9 to your shortcut specification.

Finally, please do note that specifying shortcuts using the underscore character and `Menuitem.Shortcut` are not mutually exclusive. You can actually specify both and on some systems (e.g. Windows) RapaGUI will even support both. For example, on Windows you can access menu items via the keyboard by first pressing `Alt` and then the underscore shortcut or you can also access menu items by pressing a predefined shortcut, e.g. `Ctrl+V` for paste. On systems which don't support both kinds of shortcuts, the shortcut specified in `Menuitem.Shortcut` will take precedence.

TYPE
String

APPLICABILITY
I

32.7 Menuitem.Title

NAME
Menuitem.Title – set/get menu item title

FUNCTION
Set or get the menu item title.

If the title contains an underscore, RapaGUI will automatically set up the character following this underscore as a keyboard shortcut. If you don't want this behaviour, set `Menuitem.NoAutoKey` to `True`.

TYPE
String

APPLICABILITY
ISG

32.8 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` Radio menu item that forms a group with its neighbouring radio menu items. Only one member of the group can be selected at a time.

`About` Designate item as the about menu item. Only supported on macOS. (V2.0)

`Help` Designate item as the help menu item. Only supported on macOS. (V2.0)

`Preferences`
Designate item as the preferences menu item. Only supported on macOS. (V2.0)

`Quit` Designate item as the quit menu item. Only supported on macOS. (V2.0)

Note that when setting this attribute to one of the types only supported on macOS, RapaGUI will move the item to its appropriate position according to macOS' styleguide, e.g. the `"About"`, `"Preferences"`, and `"Quit"` menus are part of the program menu on macOS. On non-macOS systems, those special types don't have any effect and RapaGUI will just ignore them.

TYPE

String (see above for possible values)

APPLICABILITY

IG

33 MOAI class

33.1 Overview

MOAI class is the superclass of all other MOAI classes. It manages the internal notification mechanism on the one hand but on the other hand it also provides some general attributes which can be used with all object types, e.g. you can store user data inside your objects using `MOAI.UserData`.

33.2 MOAI.Class

NAME
MOAI.Class – get class name of object

FUNCTION
Get the RapaGUI class name of an object.

TYPE
String

APPLICABILITY
G

33.3 MOAI.I18N

NAME
MOAI.I18N – set catalog string index (V2.0)

FUNCTION
This attribute can be used to set the catalog string index for the object. This is only supported by widgets which allow you to pass a text string as part of its XML declaration, e.g. widgets of type Button class. You have to pass the index of the catalog string for the widget in this tag, either as a numeric value or as a Hollywood constant.

See Section 3.17 [Internationalization], page 24, for details.

TYPE
Number or Hollywood constant

APPLICABILITY
I

33.4 MOAI.ID

NAME
MOAI.ID – set object ID

FUNCTION
This attribute can be used to set the ID for a MOAI object. You need to give your objects unique IDs so that you can access them using the `moai.Set()`, `moai.Get()` and `moai.DoMethod()` functions.

TYPE
String

APPLICABILITY
I

33.5 MOAI.NoNotify

NAME
MOAI.NoNotify – disable notifications

FUNCTION
When setting up notifications on class attributes, events will also be triggered in case an attribute's value is modified manually by calling `moai.Set()`. Setting `MOAI.NoNotify` in the same call will prevent the notification from being triggered.

Note that `MOAI.NoNotify` is a "one time" attribute. It's only effective during the current `moai.Set()` call!

TYPE
Boolean

APPLICABILITY
S

EXAMPLE
```
moai.Set("lv", "active", 5, "nonotify", true)
```
The code above activates listview entry number 6 but doesn't trigger any notification.

33.6 MOAI.Notify

NAME
MOAI.Notify – set up notifications

FUNCTION
Use this attribute to specify which attributes you would like to listen to. Whenever the value of the attribute changes, RapaGUI will run your event handler. Multiple attributes have to be separated by semicolons. For example, to listen to the attribute `Listview.Active` and `Listview.DoubleClick`, you would have to pass the string "active; doubleclick" here.

To set up or remove notifications outside XML declarations, use the function `moai.Notify()`. See Section 6.11 [moai.Notify], page 53, for details.

TYPE
String

APPLICABILITY
I

33.7 MOAI.NotifyData

NAME
MOAI.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 `MOAI.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 3.7 [Notifications], page 13, for details.

TYPE
Any

APPLICABILITY
ISG

33.8 MOAI.UserData

NAME
MOAI.UserData – set/get user data of object

FUNCTION
Store any kind of value in an object using this attribute. 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()`. The event handler callback will receive the user data specified in `MOAI.UserData` in the `MOAIUserData` field of the event message.
See Section 3.7 [Notifications], page 13, for details.

TYPE
Any

APPLICABILITY
ISG

34 Pageview class

34.1 Overview

Pageview class derives from Area class and is a container for multiple pages of widgets, displayed as one page at a time. When using this class, you only have to supply a number of groups constituting the pageview's children. These children can then be visualized in a number of ways, depending on the `Pageview.Mode` attribute which determines the widget that is used to browse through the pages. The following widgets are currently available for that purpose:

- tabbed widget (default)
- list widget
- choice widget
- none, i.e. pages can only be changed programmatically (useful for setup wizards etc.)

When setting up a pageview, you need to use the `Group.Title` attribute to define title texts for your pageview children. You can also use the `Group.Icon` attribute to add icons for your pageview children.

Here is an example of a three page pageview group:

```
<pageview>
   <vgroup title="Page 1">
       <listview>
          <column>
             <item>Entry</item>
          </column>
       </listview>
   </vgroup>
   <vgroup title="Page 2">
       <texteditor/>
   </vgroup>
   <vgroup title="Page 3">
       <button id="btn">Click me</button>
   </vgroup>
</pageview>
```

34.2 Pageview.Active

NAME

Pageview.Active – set/get active page

FUNCTION

Set or get the active page. The pageview widget will only display this page, all other pages will be hidden. Page indices range from 0 for the first page to `Pageview.Pages`-1 for the last child.

You can also pass one of the following special values here:

`First` First page.

`Last` Last page.

`Prev` Previous page.

`Next` Next page.

You can also set up a notification on this attribute to learn when the user changes pages.

TYPE

Number or string (see above for possible values)

APPLICABILITY

ISGN

34.3 Pageview.Append

NAME

Pageview.Append – add new page to pageview

SYNOPSIS

```
moai.DoMethod(id, "Append", obj[, active])
```

FUNCTION

This method can be used to add a new page to a pageview. The new page specified by `obj` must be a detached group object. This detached group object will then be added as the pageview'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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Pageview.Remove` method.

On AmigaOS and compatibles this feature requires at least MUI 4.0.

INPUTS

`id` id of the pageview object

`obj` id of the group object to add as a new page

`active` optional: set this to `True` to activate the page after adding it

34.4 Pageview.GetPageID

NAME

Pageview.GetPageID – return ID of a page inside the pageview

SYNOPSIS

```
id$ = moai.DoMethod(id, "GetPageID", idx)
```

FUNCTION

This method returns the ID of the page at the specified index within the pageview. `idx` can be an absolute number ranging from 0 to the number of pages in the pageview minus 1 or it can be one of the following special values:

`First` First page.

`Last` Last page.

`Active` Active page.

On AmigaOS and compatibles this feature requires at least MUI 4.0.

INPUTS

`id` id of the pageview object

`idx` absolute index of desired page or special string constant (see above)

RESULTS

`id$` id of the page object at the specified pageview index

34.5 Pageview.Insert

NAME

Pageview.Insert – insert new page into pageview

SYNOPSIS

```
moai.DoMethod(id, "Insert", obj, pos[, active])
```

FUNCTION

This method can be used to insert a new page into a pageview at the position specified by `pos`. Insert positions are counted from 0 which marks the position of the first page. The new page specified by `obj` must be a detached group object. 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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Pageview.Remove` method.

On AmigaOS and compatibles this feature requires at least MUI 4.0.

INPUTS

`id` id of the pageview object

`obj` id of the group object to insert as a new page

`pos` desired insert position for the new page (starting from 0)

`active` optional: set this to `True` to activate the page after adding it

34.6 Pageview.Mode

NAME

Pageview.Mode – set browser widget to use

FUNCTION

Set the browser widget this pageview object should use. This can be one of the following values:

`Tabs` Use a tabbed control to browse through the pages. This is the default mode.

`List` Use a listview widget to browse through the pages.

`Choice` Use a choice widget to browse through the pages.

`None` Don't use any browser widget at all. This means that the user cannot browse through the pages. Pages can only be changed by setting the `Pageview.Active` attribute. This is useful for setup or installation wizards which often use a custom pageview which only switches page when the user presses a "Next page" or "Previous page" button.

TYPE
String (see above for possible values)

APPLICABILITY
I

34.7 Pageview.Multiline

NAME
Pageview.Multiline – enable multi line tabs

PLATFORMS
Windows only

FUNCTION
Set this to `True` to make the browser widget show its tabs on multiple lines instead of adding scroll buttons when the number of tabs exceeds the available space. Obviously, this attribute only makes sense when using `Tabs` mode with `Pageview.Mode`.

TYPE
Boolean

APPLICABILITY
I

34.8 Pageview.Pages

NAME
Pageview.Pages – find out number of pages in pageview

FUNCTION
You can query this attribute to find out the current number of pages in the pageview object.

On AmigaOS and compatibles this feature requires at least MUI 4.0.

TYPE
Number

APPLICABILITY
G

34.9 Pageview.PlainBG

NAME
Pageview.PlainBG – force normal group background

PLATFORMS
AmigaOS and compatibles only

FUNCTION
By default the individual pageview children groups are drawn with a special background color to indicate that they are pageview children. Set this attribute to `True` if you don't want this special background.

TYPE
Boolean

APPLICABILITY
I

34.10 Pageview.Position

NAME
Pageview.Position – set position of browser widget

FUNCTION
This attribute allows you to configure where the pageview's browser widget (register, list, or choice widget) should appear.

The following values are recognized by this attribute:

`Left` Put browser widget to the left of the pages.

`Right` Put browser widget to the right of the pages.

`Bottom` Put browser widget to the bottom of the pages.

`Top` Put browser widget to the top of the pages.

On AmigaOS and compatibles this feature requires at least MUI 4.0.

TYPE
String (see above for possible values)

APPLICABILITY
I

34.11 Pageview.Prepend

NAME
Pageview.Prepend – prepend new page to pageview

SYNOPSIS
```
moai.DoMethod(id, "Prepend", obj[, active])
```

FUNCTION

This method can be used to prepend a new page to a pageview. The new page specified by `obj` must be a detached group object. This detached group object will then be added as the pageview's 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 MOAI objects can be created either by calling the `moai.CreateObject()` function or by explicitly detaching them from their parent by using the `Pageview.Remove` method.

On AmigaOS and compatibles this feature requires at least MUI 4.0.

INPUTS

`id`	id of the pageview object
`obj`	id of the group object to add as the first page
`active`	optional: set this to `True` to activate the page after adding it

34.12 Pageview.Remove

NAME

Pageview.Remove – remove page from pageview

SYNOPSIS

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

FUNCTION

This method will remove the page specified by `obj` from the pageview. The group object that constitutes the page that is being removed will then change its state from attached to detached. This means that you could attach it now to another pageview or group using a function like `Group.Insert` and `Pageview.Append`, respectively, or you could free it using `moai.FreeObject()`.

On AmigaOS and compatibles this feature requires at least MUI 4.0.

INPUTS

`id`	id of the pageview object
`obj`	id of the page to remove

35 Popcolor class

35.1 Overview

Popcolor class derives from Area class and creates a widget which allows the user to pick a color. Popcolor widgets usually contain a button which, when clicked, pops up a dialog prompting the user to choose a color. This is all managed automatically by the widget and you don't have to do anything except putting the widget somewhere in your window layout.

You can set the title of the popup dialog using the `Popcolor.Title` attribute on the Popcolor object. To get or set the current color of a Popcolor object, you can use the `Popcolor.RGB` attribute.

35.2 Popcolor.RGB

NAME
Popcolor.RGB – set/get color

FUNCTION
Set or get the popcolor 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

35.3 Popcolor.Title

NAME
Popcolor.Title – set popup window title

FUNCTION
Set the title for the popup dialog.

TYPE
String

APPLICABILITY
I

36 Popfile class

36.1 Overview

Popfile class derives from Area class and creates a widget which allows the user to select a file. Popfile widgets usually contain a button which, when clicked, pops up a dialog prompting the user to choose a file. This is all managed automatically by the widget and you don't have to do anything except putting the widget somewhere in your window layout.

You can set the title of the popup dialog using the `Popfile.Title` attribute on the Popfile object and you can get the user's selection by getting the `Popfile.File` attribute.

36.2 Popfile.File

NAME
Popfile.File – set/get current popfile path

FUNCTION
Get and set the current path of this popfile object.

You can also set up a notification on this attribute to be notified whenever the user selects a new file for this popfile object.

TYPE
String

APPLICABILITY
ISGN

36.3 Popfile.Pattern

NAME
Popfile.Pattern – set filter pattern for file requester

FUNCTION
Set the filter pattern for the file requester. Only files which match this filter pattern will be selectable in the requester. The filter pattern is a string that contains a number of file extensions that will be accepted by the requester. These endings must be separated by '|' characters. For example: `"voc|wav|8svx|16sv|iff|aiff"` will only show files which have one of these extensions.

If you don't set this attribute, all files will be selectable.

TYPE
String

APPLICABILITY
I

36.4 Popfile.SaveMode

NAME
Popfile.SaveMode – enable save mode

FUNCTION
Set this tag to `True` to put the file requester in save mode.

TYPE
Boolean

APPLICABILITY
I

36.5 Popfile.Title

NAME
Popfile.Title – set requester title

FUNCTION
Set the title string for the file requester window.

TYPE
String

APPLICABILITY
I

37 Popfont class

37.1 Overview

Popfont class derives from Area class and creates a widget which allows the user to select a font. Popfont widgets usually contain a button which, when clicked, pops up a dialog prompting the user to choose a font. This is all managed automatically by the widget and you don't have to do anything except putting the widget somewhere in your window layout.

You can set the title of the popup dialog using the `Popfont.Title` attribute on the Popfont object and you can get the user's selection by getting the `Popfont.Font` attribute.

37.2 Popfont.Font

NAME
Popfont.Font – set/get current font selection

FUNCTION
Get and set the user's current font selection. Popfont class uses a slash to separate font name and size, e.g. `Arial/48`.

You can also set up a notification on this attribute to be notified whenever the user selects a new font for this popfont object.

TYPE
String

APPLICABILITY
ISGN

37.3 Popfont.MaxSize

NAME
Popfont.MaxSize – set maximum font height

FUNCTION
Only display fonts which are equal to or smaller than this size.

TYPE
Number

APPLICABILITY
I

37.4 Popfont.MinSize

NAME
Popfont.MinSize – set minimum font height

FUNCTION
Only display fonts which are equal to or larger than this size.

TYPE
Number

APPLICABILITY
I

37.5 Popfont.Title

NAME
Popfont.Title – set requester title

FUNCTION
Set the title string for the font requester window.

TYPE
String

APPLICABILITY
I

38 Poppath class

38.1 Overview

Poppath class derives from Area class and creates a widget which allows the user to select a path. Poppath widgets usually contain a button which, when clicked, pops up a dialog prompting the user to choose a path. This is all managed automatically by the widget and you don't have to do anything except putting the widget somewhere in your window layout.

You can set the title of the popup dialog using the `Poppath.Title` attribute on the Poppath object and you can get the user's selection by getting the `Poppath.Path` attribute.

38.2 Poppath.Path

NAME
Poppath.Path – set/get current poppath path

FUNCTION
Get and set the current path of this poppath object.

You can also set up a notification on this attribute to be notified whenever the user selects a new path for this poppath object.

TYPE
String

APPLICABILITY
ISGN

38.3 Poppath.Title

NAME
Poppath.Title – set requester title

FUNCTION
Set the title string for the requester window.

TYPE
String

APPLICABILITY
I

39 Progressbar class

39.1 Overview

Progressbar class derives from Area class and creates a widget which visualizes tasks or processes that take some time to complete. Usually, the rest of the application is blocked while a progress bar is being shown.

Here is an example XML declaration of a progress bar in RapaGUI:

```
<progressbar/>
```

39.2 Progressbar.Horiz

NAME
Progressbar.Horiz – set progress bar orientation

FUNCTION
Set this to `False` if you want a vertical progress bar. By default, a horizontal progress bar will be created.

TYPE
Boolean

APPLICABILITY
I

39.3 Progressbar.Level

NAME
Progressbar.Level – set/get current level

FUNCTION
Set the current position of the progress bar. The new position must be between 0 and `Progressbar.Max`. You can also set up a notification on this attribute. This can be useful if you'd like to update some other widget whenever the current level changes.

TYPE
Number

APPLICABILITY
ISGN

39.4 Progressbar.Max

NAME
Progressbar.Max – set maximum level of progress bar

FUNCTION
Set the maximum level for the progressbar. This defaults to 100.

TYPE
Number

APPLICABILITY
ISG

40 Radio class

40.1 Overview

Radio class derives from Area class and creates a number of mutually exclusive buttons allowing the user to make a single choice. The buttons are embedded inside a group with a frame and an optional frame title.

When you declare a radio widget in XML, you have to use the `<item>` tag to fill the radio widget with items. Every radio widget needs to have at least one item.

Here is an example XML excerpt for creating a radio widget:

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

You can use the `Radio.Columns` attribute to fine-tune the layout of the radio buttons. By default, they appear in a single-column vertical layout. Setting `Radio.Columns` allows you to change that to a horizontal layout, for example.

40.2 Radio.Active

NAME

Radio.Active – set/get active radio item

FUNCTION

Set or get the active item in the radio widget ranging from index 0 for the first item to number of entries - 1 for the last item.

You can also set up a notification on this attribute to learn whenever the user selects a new radio item.

TYPE

Number

APPLICABILITY

ISGN

40.3 Radio.Columns

NAME

Radio.Columns – set number of radio columns (V2.0)

FUNCTION

Set the number of columns the radio widget should use. This defaults to 1 which means that all radio buttons will appear in one column.

Note that on Android, you cannot have complex radio layouts that use columns and rows. You can either have all radio buttons appear in one row (horizontal layout) or in one column (vertical layout).

TYPE
Number

APPLICABILITY
I

40.4 Radio.GetItem

NAME
Radio.GetItem – get radio button label (V2.0)

SYNOPSIS
```
e$ = moai.DoMethod(id, "GetItem", pos)
```

FUNCTION
Get the label of a radio button. You can pass either an absolute index in `pos` or the special value `Active` to get the label of the active button. `Radio.GetItem` will then return the label as a string.

INPUTS

`id`	id of the radio object
`pos`	button index or "Active"

RESULTS

`e$`	label of radio button at specified index

40.5 Radio.SetItem

NAME
Radio.SetItem – set radio button label (V2.0)

SYNOPSIS
```
moai.DoMethod(id, "SetItem", pos, newname$)
```

FUNCTION
Set the radio button label to the name specified in `newname$`. The index of the radio button to use is specified in the `pos` argument. This can be an absolute index position starting at 0 for the first button or one of the following special values:

`Active`	Change the label of the active button.

INPUTS

`id`	id of the radio object
`pos`	button position as absolute number or special value (see above)
`newname$`	new label text

40.6 Radio.Title

NAME

Radio.Title – set title for radio box frame

FUNCTION

The text you specify here is shown as a title text in the frame that is drawn around the radio box.

Note that changing the frame title requires MUI 4.0 or better on AmigaOS and compatibles.

TYPE

String

APPLICABILITY

ISG

41 Rectangle class

41.1 Overview

Rectangle class derives from Area class and creates empty objects that are freely resizable. This might not seem very useful at first but in fact, rectangle objects are needed very often to fine-tune the GUI layout. They can be used for all kinds of layout magic, e.g. for controlling the alignment of fixed-size widgets and much more.

Additionally, rectangle objects are often necessary as padding space next to objects which are not resizable themselves. The `<radio>` object, for instance, is not resizable. To prevent these static radio objects from blocking the resizing feature of your whole GUI, you can simply pad them with `<rectangle>` objects. Here is an example of a radio object, padded using an invisible rectangle:

```
<hgroup>
  <rectangle/>
  <radio id="printer">
    <item>HP Deskjet</item>
    <item>NEC P6</item>
    <item>Okimate 20</item>
  </radio>
   <rectangle/>
</hgroup>
```

By using rectangles on both sides of the radio box object, the radio box will automatically be centered inside the available GUI space. If you used only one rectangle object before the radio object, the radio object would automatically be right aligned. A rectangle object after the radio object would lead to left alignment of the radio object.

If you need fixed size padding objects, use HSpace class or VSpace class. See Section 21.1 [HSpace class], page 123, for details. See Section 60.1 [VSpace class], page 295, for details.

Rectangle class doesn't define any attributes.

42 Scrollbar class

42.1 Overview

Scrollbar class derives from Area class and creates a widget that represents a horizontal or vertical scrollbar. An isolated scrollbar object doesn't make much sense. That's why scrollbar objects are typically connected to another widget by using the `Scrollbar.Target` attribute. For example, you can connect scrollbars to an object of Hollywood class to create a custom widget which draws user-defined graphics depending on the current scrollbar position.

42.2 Scrollbar.AutoScale

NAME
Scrollbar.AutoScale – enable/disable autoscaling (V2.0)

FUNCTION
By default, all size and position values in scrollbar class will use device-independent pixels. If you don't want that, set this attribute to `False`. In that case, all values will use physical pixels instead. See Section 3.12 [High-DPI support], page 20, for details.

This attribute defaults to what has been set in the `ScaleGUI` tag when calling `@REQUIRE` on the RapaGUI plugin. Note that when not set, `ScaleGUI` defaults to `True`. See Section 3.4 [Initializing RapaGUI], page 11, for details.

TYPE
Boolean

APPLICABILITY
I

42.3 Scrollbar.Horiz

NAME
Scrollbar.Horiz – set scrollbar object direction

FUNCTION
Determine whether you want a horizontal or a vertical scrollbar. By default, a vertical scrollbar is created.

TYPE
Boolean

APPLICABILITY
IG

42.4 Scrollbar.Level

NAME
Scrollbar.Level – set/get current scrollbar position

FUNCTION
Set or get the current scrollbar position. You usually will want to set up a notification on this attribute to be notified whenever the scrollbar position changes. This allows you to dynamically react to these changes and update another widget which your scrollbar is determined to control.

Note that unless `Scrollbar.AutoScale` is to `False`, this value will be in device-independent pixels. If you want to have physical pixels instead, you must set `Scrollbar.AutoScale` to `False`.

TYPE
Number

APPLICABILITY
ISGN

42.5 Scrollbar.Range

NAME
Scrollbar.Range – set/get scrollbar range

FUNCTION
Set or get the scrollbar range.

Note that unless `Scrollbar.AutoScale` is to `False`, this value will be in device-independent pixels. If you want to have physical pixels instead, you must set `Scrollbar.AutoScale` to `False`.

TYPE
Number

APPLICABILITY
ISG

42.6 Scrollbar.StepSize

NAME
Scrollbar.StepSize – set/get scrollbar step size

FUNCTION
Set or get the number of pixels to scroll when the user clicks on one of the scrollbar's buttons.

Note that unless `Scrollbar.AutoScale` is to `False`, this value will be in device-independent pixels. If you want to have physical pixels instead, you must set `Scrollbar.AutoScale` to `False`.

TYPE
Number

APPLICABILITY
ISG

42.7 Scrollbar.Target

NAME
Scrollbar.Target – set target widget for scrollbar

FUNCTION
Use this attribute to define the widget that is to be controlled by this scrollbar object. This should always be provided because the scrollbar's thumb size changes whenever the size of its target widget changes. If you specify a target widget here, RapaGUI will automatically update the scrollbar's thumb size whenever the size of the target widget changes. Otherwise you will have to do this manually by setting `Scrollbar.Visible` whenever the thumb size changes.

TYPE
MOAI object

APPLICABILITY
I

42.8 Scrollbar.UseWinBorder

NAME
Scrollbar.UseWinBorder – put scrollbar object into window border

PLATFORMS
AmigaOS and compatibles only

FUNCTION
Set this attribute to make RapaGUI put the scrollbar object into the window border instead of creating a widget which can be put into a group object.

Before using `Scrollbar.UseWinBorder` you first have to enable border scrollers for the parent window by setting the appropriate window class attributes. For example, if you want to put the scrollbar into the bottom window border, you would have to set the `Window.UseBottomBorderScroller` attribute first. See Section 62.1 [Window class], page 301, for details.

Also note that there obviously can only be two scrollbars in the window border: One on the left or right, and one in the bottom window border.

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

42.9 Scrollbar.Visible

NAME
Scrollbar.Visible – set/get number of visible entries

FUNCTION
Set or get the number of visible entries and update the scrollbar's thumb size accordingly. If you haven't provided a target widget by specifying `Scrollbar.Target`, you have to update the scrollbar's thumb size using this attribute whenever the size of the target widget changes.

Note that unless `Scrollbar.AutoScale` is to `False`, this value will be in device-independent pixels. If you want to have physical pixels instead, you must set `Scrollbar.AutoScale` to `False`.

TYPE
Number

APPLICABILITY
ISG

43 Scrollcanvas class

43.1 Overview

Scrollcanvas class derives from Area class and creates a canvas with scrollbars attached. You can draw custom graphics to this canvas via a paint callback that is automatically invoked whenever content needs to be drawn. You just need to set up a notification on the `Scrollcanvas.Paint` attribute and your event handler will be called whenever content needs to be drawn. The same thing can be achieved by embedding a widget derived from Hollywood class inside a scrollgroup or connecting such a Hollywood widget to scrollbars but Scrollcanvas class is preferable in some cases because it is optimized especially for scrolled content, i.e. it tries to minimize painting by using OS widgets which are specifically designed for displaying scrolled content. Thus, it is usually faster than the two solutions outlined above.

You can set the canvas dimensions by using the `Scrollcanvas.VirtWidth` and `Scrollcanvas.VirtHeight`. Since Scrollcanvas class uses scrollbars, the dimensions of the canvas can obviously be much bigger than the physical dimensions of the scrollcanvas widget. As with all other classes, you can set those physical dimensions using the generic `Area.Width` and `Area.Height` attributes.

To force a complete redraw of your widget, just run the `Area.Redraw` method on your object. Running `Area.Redraw` on your object will result in your paint function being called so that you can update the canvas accordingly.

With Scrollcanvas class the dimensions you specify in `Scrollcanvas.VirtWidth` and `Scrollcanvas.VirtHeight` are in device-independent pixels by default and RapaGUI will automatically apply the system's scale factor to the contents drawn by the `Scrollcanvas.Paint` function. If you want to have fine-tuned control, you can set the `Scrollcanvas.AutoScale` attribute to `False`. In that case, `Scrollcanvas.VirtWidth` and `Scrollcanvas.VirtHeight` are interpreted as physical pixels and no auto scaling will be done so that your paint function can draw high resolutions graphics without any quality loss due to scaling. See Section 3.12 [High-DPI support], page 20, for details.

43.2 Scrollcanvas.AutoBars

NAME

Scrollcanvas.AutoBars – set scrollbar visibility

FUNCTION

By default, the scrollbars will automatically be hidden when they are not needed. If you don't want this behaviour, set this attribute to `False`. In that case the scrollbars will always be visible even when they're not needed.

TYPE

Boolean

APPLICABILITY

I

43.3 Scrollcanvas.AutoScale

NAME

Scrollcanvas.AutoScale – enable/disable autoscaling (V2.0)

FUNCTION

By default, scrollcanvas class will use device-independent pixels and will then scale its canvas to fit to the current monitor's DPI settings. If you don't want that, set this attribute to `False`. In that case, your scrollcanvas widget will operate completely in physical pixel mode and no autoscaling will be done. See Section 3.12 [High-DPI support], page 20, for details.

This attribute defaults to what has been set in the `ScaleGUI` tag when calling `@REQUIRE` on the RapaGUI plugin. Note that when not set, `ScaleGUI` defaults to `True`. See Section 3.4 [Initializing RapaGUI], page 11, for details.

TYPE

Boolean

APPLICABILITY

I

43.4 Scrollcanvas.Paint

NAME

Scrollcanvas.Paint – request paint notification

FUNCTION

Set up a listener on this attribute to have your event handler called whenever content needs to be drawn on the canvas. Your event handler can then paint the canvas contents depending on the current scrollbar position.

RapaGUI will pass the identifier of a Hollywood brush whose size is exactly as big as the visible area of your scrollcanvas widget to your callback. You then have to draw the desired contents to this brush. Precisely, you just have to draw to the rectangle defined by the four coordinates `X`, `Y`, `Width`, and `Height` which are passed to your callback as well. These four coordinates describe a rectangular area within the dimensions of the brush that is passed to your callback. When a full redraw is needed, `X` and `Y` will be 0 and `Width` and `Height` will match the dimensions of the brush. Most oftenly, however, only a partial redraw is needed and then you must only draw to the portion of the brush defined by those coordinates.

The following extra arguments will be passed to your event handler:

`Brush:` Contains the identifier of a brush you have to draw to. Use Hollywood's `SelectBrush()` command to select this brush as the output device in your callback. Don't forget to call `EndSelect()` when you are done!

`ViewWidth:`
Contains the visible width of widget. This is also identical to the width of the brush that is passed to your callback.

`ViewHeight:`
: Contains the visible height of widget. This is also identical to the height of the brush that is passed to your callback.

`ScrollX:` Contains the position of the horizontal scrollbar.

`ScrollY:` Contains the position of the vertical scrollbar.

`VirtWidth:`
: Contains the virtual width of your widget. This is the value set using `Scrollcanvas.VirtWidth`.

`VirtHeight:`
: Contains the virtual height of your widget. This is the value set using `Scrollcanvas.VirtHeight`.

`X:` Contains the x-position inside the brush at which you should start drawing. See above for details.

`Y:` Contains the y-position inside the brush at which you should start drawing. See above for details.

`Width:` Contains the number of columns you should paint to the brush (starting from `X`). See above for details.

`Height:` Contains the number of rows you should paint to the brush (starting from `Y`). See above for details.

To compute the absolute position of the content that should be drawn to the canvas, just add the `ScrollX+X` and `ScrollY+Y` coordinates and you're done.

See Section 3.7 [Notifications], page 13, for details.

TYPE
Boolean

APPLICABILITY
N

43.5 Scrollcanvas.Scroll

NAME
Scrollcanvas.Scroll – scroll the canvas

SYNOPSIS
```
moai.DoMethod(id, "Scroll", x, y)
```

FUNCTION
Scrolls the canvas to the positions specified by `x` and `y`.

INPUTS

`id` id of the scrollcanvas object

`x` desired new x position

`y` desired new y position

43.6 Scrollcanvas.StepSize

NAME
Scrollcanvas.StepSize – set/get scrollbar step size

FUNCTION
Set or get the number of pixels to scroll when the user clicks on one of the scrollbar's buttons.

TYPE
Number

APPLICABILITY
ISG

43.7 Scrollcanvas.UseLeftBorder

NAME
Scrollcanvas.UseLeftBorder – use left border scroller

PLATFORMS
AmigaOS and compatibles only

FUNCTION
When using `Scrollcanvas.UseWinBorder` the scrollbars will be put into the right and bottom window borders by default. If you want to have the vertical scrollbar in the left window border instead, set the `Scrollcanvas.UseLeftBorder` attribute.

Don't forget to set the `Window.UseLeftBorderScroller` attribute as well.

TYPE
Boolean

APPLICABILITY
I

43.8 Scrollcanvas.UseWinBorder

NAME
Scrollcanvas.UseWinBorder – use window scrollbars for this scrollcanvas

PLATFORMS
AmigaOS and compatibles only

FUNCTION
Set this attribute to make RapaGUI put the scrollcanvas' scrollbars into the window border instead of creating them as normal widgets.

By default, the scrollbars will be put into the right and bottom window borders. If you want to have the vertical scrollbar in the left window border instead, set the `Scrollcanvas.UseLeftBorder` attribute.

Before using `Scrollcanvas.UseWinBorder` you first have to enable border scrollers for the parent window by setting the appropriate window class attributes. For the standard configuration (right and bottom border scrollers), you would have to set the `Window.UseBottomBorderScroller` and `Window.UseRightBorderScroller` attributes. See Section 62.1 [Window class], page 301, for details.

TYPE
Boolean

APPLICABILITY
I

43.9 Scrollcanvas.VirtHeight

NAME
Scrollcanvas.VirtHeight – set/get canvas height

FUNCTION
Sets or gets the canvas height. Normally this value must be in device-independent pixels. If you want your scrollcanvas to use physical pixels instead of device-independent ones, you must set the `Scrollcanvas.AutoScale` attribute to `False`. See Section 3.12 [High-DPI support], page 20, for details.

This is called "virtual height" because usually only a portion of it is actually visible. The user can use the scrollbars to scroll through the complete canvas content.

This value must be provided.

TYPE
Number

APPLICABILITY
ISG

43.10 Scrollcanvas.VirtWidth

NAME
Scrollcanvas.VirtWidth – set/get canvas width

FUNCTION
Sets or gets the canvas width. Normally this value must be in device-independent pixels. If you want your scrollcanvas to use physical pixels instead of device-independent ones, you must set the `Scrollcanvas.AutoScale` attribute to `False`. See Section 3.12 [High-DPI support], page 20, for details.

This is called "virtual width" because usually only a portion of it is actually visible. The user can use the scrollbars to scroll through the complete canvas content.

This value must be provided.

TYPE
Number

APPLICABILITY
ISG

44 Scrollgroup class

44.1 Overview

Scrollgroup class is a special variant of Group class which adds scrollbars to groups to make it possible to use groups which are larger than the available GUI space. The user can simply scroll through the group then using the scrollbars attached to the group. When there is enough space for the complete group, the scrollbars are automatically hidden unless explicitly configured not to do so.

Scrollgroups can be created in a similar fashion to normal groups. Here is an XML example:

```
<scrollgroup>
    <radio>
       <item>Amiga 500</item>
       <item>Amiga 1200</item>
       <item>Amiga 4000</item>
    </radio>
    <listview>
       <column/>
    </listview>
</scrollgroup>
```

The XML code above embeds a radio and a listview object inside a scrollgroup.

44.2 Scrollgroup.AutoBars

NAME
Scrollgroup.AutoBars – set scrollbar visibility

FUNCTION
By default, the scrollbars will automatically be hidden when they are not needed. If you don't want this behaviour, set this attribute to `False`. In that case the scrollbars will always be visible even when they're not needed.

TYPE
Boolean

APPLICABILITY
I

44.3 Scrollgroup.Horiz

NAME
Scrollgroup.Horiz – set group orientation

FUNCTION
Set this to `True` to create a horizontal scrollgroup. By default, scrollgroups use vertical orientation.

TYPE
Boolean

APPLICABILITY
I

44.4 Scrollgroup.UseWinBorder

NAME
Scrollgroup.UseWinBorder – use window scrollbars for this scrollgroup

PLATFORMS
AmigaOS and compatibles only

FUNCTION
Set this attribute to make RapaGUI put the scrollgroup's scrollbars into the window border instead of creating them as normal widgets.

Before using `Scrollgroup.UseWinBorder` you first have to enable border scrollers for the parent window by setting the appropriate window class attributes. For example, if you want to put a scrollbar into the bottom window border, you would have to set the `Window.UseBottomBorderScroller` attribute first. See Section 62.1 [Window class], page 301, for details.

TYPE
Boolean

APPLICABILITY
I

45 Slider class

45.1 Overview

Slider class derives from Area class and creates a widget which allows the user to adjust a numeric value using a thumb which can be pulled back and forth to change the value. Sliders can use horizontal or vertical orientation depending on the `Slider.Horiz` attribute. By default a horizontal slider is created.

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"/>
```

45.2 Slider.Drag

NAME
Slider.Drag – learn about thumb drag events (V2.0)

FUNCTION
This attribute is set whenever the user starts to drag the thumb. You can set up a notification on this attribute in order to be notified whenever the user starts dragging the slider's thumb.

TYPE
Boolean

APPLICABILITY
N

45.3 Slider.Horiz

NAME
Slider.Horiz – set/get orientation of slider

FUNCTION
Specify whether you want a horizontal or vertical slider. Defaults to `True` which means create a horizontal slider.

TYPE
Boolean

APPLICABILITY
I

45.4 Slider.Level

NAME
Slider.Level – set/get current slider level

FUNCTION
The current position of the slider thumb. This value will always be between `Slider.Min` and `Slider.Max`.

You can also set up a notification on this attribute to get notified whenever the thumb position changes.

TYPE
Number

APPLICABILITY
ISGN

45.5 Slider.Max

NAME
Slider.Max – set/get maximum level of slider

FUNCTION
Set or get the maximum value for the slider object.

TYPE
Number

APPLICABILITY
ISG

45.6 Slider.Min

NAME
Slider.Min – set/get minimum level of slider

FUNCTION
Set or get the minimum value for the slider object. This can also be less than 0.

TYPE
Number

APPLICABILITY
ISG

45.7 Slider.Quiet

NAME
Slider.Quiet – do not show current slider level

FUNCTION
Set this to `True` to make the slider hide its current level. Normally, the current level is shown either inside the slider's thumb or in a text widget next to the slider.

TYPE
Boolean

APPLICABILITY
I

45.8 Slider.Release

NAME
Slider.Release – learn about thumb release events

FUNCTION
This attribute is set whenever the user releases the thumb. You can set up a notification on this attribute in order to be notified whenever the user lets loose of the thumb.

TYPE
Boolean

APPLICABILITY
N

45.9 Slider.Reverse

NAME
Slider.Reverse – reverse direction of slider

FUNCTION
Set this attribute to `True` to reverse the direction of the slider.

TYPE
Boolean

APPLICABILITY
I

45.10 Slider.StepSize

NAME
Slider.StepSize – set/get slider step size (V2.0)

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set or get the slider step size. This value indicates how much the slider's level should be increased or decreased when clicking outside the slider thumb. The default step size is 1.

TYPE
Number

APPLICABILITY
ISG

46 Statusbar class

46.1 Overview

Statusbar class derives from Area class and creates a widget placed at the bottom of the window to give small amounts of status information. Status bars contain one or more children of Statusbaritem class, all of which can be either of fixed or variable lengths.

When creating a status bar in XML, you have to specify how many fields it should get by using the `<item>` tag to add children of Statusbaritem class to your status bar. Here is an example of a status bar with three fields:

```
<statusbar>
   <item id="first">Welcome to my application</item>
   <item id="second"/>
   <item id="third" width="20"/>
</statusbar>
```

You can set the widths of the individual fields using the `Statusbaritem.Width` attribute. Above we assign a fixed width only to the third field. The other two use variable widths.

The text of status bar items can be changed later by setting the `Statusbaritem.Text` attribute. Note, however, that this often isn't even necessary because the text provided in the `Menuitem.Help` and `Toolbarbutton.Help` attributes is automatically shown in the status bar and doesn't require any additional code on your part.

There can be only one status bar per window and it always has to be the last child of the window's root group. RapaGUI won't accept status bars at random positions in the GUI layout. Status bars must always be the last child of the window's root group. Thus, they are always placed at the bottom of the window layout.

Also note that it is not possible to create stand-alone instances of this class using `moai.CreateObject()`. Status bars always have to be created within the context of a window. Thus, if you want to create status bars using `moai.CreateObject()`, you always have to create a complete window and embed the status bar in this window declaration.

Statusbar class doesn't define any attributes or methods itself. See Section 47.1 [Statusbaritem class], page 229, for all necessary information.

47 Statusbaritem class

47.1 Overview

Statusbaritem class is used to create text fields which are used as children for Statusbar class. You cannot create independent instances of Statusbaritem class. They always need to be embedded into Statusbar class. See Section 46.1 [Statusbar class], page 227, for details.

47.2 Statusbaritem.Text

NAME
Statusbaritem.Text – set/get item text

FUNCTION
Set or get the status bar item's text.

TYPE
String

APPLICABILITY
SG

47.3 Statusbaritem.Width

NAME
Statusbaritem.Width – set item width

FUNCTION
Set the desired width of the status bar item. This can be either an absolute value in device-independent pixels for a fixed width field or a negative value indicating a proportion for a variable width field. The space left for all variable width fields is divided between them according to the absolute value of this number. A variable width field with width of -2 gets twice as much of it as a field with width -1 and so on.

For example, to create one fixed width field of width 100 in the right part of the status bar and two more fields which get 66% and 33% of the remaining space correspondingly, you would set the width fields of the items to -2, -1 and 100, respectively.

Defaults to -1.

TYPE
Number

APPLICABILITY
I

48 Text class

48.1 Overview

Text class derives from Area class and creates widgets that display one or more lines of text, pretty similar to what widgets of Label class do. A difference is that Text class supports multiple lines of text as well as frames around the text and widgets created by Text class are resizable whereas Label class creates non-sizeable widgets. Like Label class, Text class also supports the underscore character in its text to define a keyboard shortcut.

Please note that Text class supports neither automatic word wrapping nor text formatting. If you want to have these features, you have to use Textview class instead. See Section 51.1 [Textview class], page 255, for details.

Here is an example of how to use the `<text>` tag in XML:

```
<text>Hello World</text>
```

48.2 Text.Align

NAME
Text.Align – set/get text alignment

FUNCTION
Set the desired alignment for the text. This can be one of the following values:

`Left` Left alignment. This is the default.

`Right` Right alignment.

`Center` Centered alignment.

TYPE
String (see above for possible values)

APPLICABILITY
ISG

48.3 Text.Frame

NAME
Text.Frame – add frame around text widget

FUNCTION
Set this to `True` to get a frame around the text widget.

TYPE
Boolean

APPLICABILITY
I

48.4 Text.Text

NAME
Text.Text – set/get text string

FUNCTION
Set or get the text string. The text may contain newline characters.

TYPE
String

APPLICABILITY
SG

49 Texteditor class

49.1 Overview

Texteditor class derives from Area class and creates multi-line text entry widgets with most of the functions of a normal text editor. It also supports text formatting via certain character codes if `Texteditor.Styled` has been set. This allows you to enable certain styles (bold, italic, underline) for your text as well as change the color of your text. See Section 3.14 [Text formatting codes], page 22, for details. The XML tag's content is used as the initial contents of the text editor widget.

Here is an XML example of how to include a text editor object in your GUI:

```
<texteditor>Enter your text here!</texteditor>
```

Note that there is a slight difference between the texteditor widget on Windows and all other platforms: Windows' native texteditor widget uses two characters between lines (CRLF, i.e. carriage return and linefeed) whereas on all other platforms only a linefeed character is used. This leads to the problem that hard-coded index or range positions are not completely portable because Windows always uses two characters to start a new line. Be prepared to deal with this problem.

Also note that this class requires the `TextEditor.mcc` extension to be installed on AmigaOS and compatibles.

49.2 Texteditor.Align

NAME
Texteditor.Align – set/get text alignment

FUNCTION
Set/get the text alignment.

The following values are possible:

`Left` Left alignment.

`Right` Right alignment.

`Center` Centered alignment.

TYPE
String (see above for possible values)

APPLICABILITY
ISG

49.3 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

49.4 Texteditor.Bold

NAME
Texteditor.Bold – 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.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

TYPE
Boolean

APPLICABILITY
SGN

49.5 Texteditor.Clear

NAME
Texteditor.Clear – clear text

SYNOPSIS
```
moai.DoMethod(id, "Clear")
```

FUNCTION
This will clear all the text in the widget.

Note that even though setting this attribute will not trigger a `Texteditor.HasChanged` notification it will set `Texteditor.HasChanged` to `False`.

INPUTS

`id`	id of the text editor object

49.6 Texteditor.Color

NAME
Texteditor.Color – set/get current text color

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. The color has to be passed as a 24-bit RGB value.

You can also setup a notification on this attribute to learn when the cursor has been moved over text in a different color.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

TYPE
Number

APPLICABILITY
SGN

49.7 Texteditor.Copy

NAME
Texteditor.Copy – copy marked text

SYNOPSIS
```
moai.DoMethod(id, "Copy")
```

FUNCTION
Copy currently selected text to clipboard.

INPUTS
`id` id of the text editor object

49.8 Texteditor.CursorPos

NAME
Texteditor.CursorPos – set/get cursor position

FUNCTION
You can get or set the cursor's position with this tag. The initial character starts at position 0.

You can also set up a notification on this tag to learn when the cursor is moved.

TYPE
Number

APPLICABILITY
SGN

49.9 Texteditor.Cut

NAME
Texteditor.Cut – cut marked text

SYNOPSIS
```
moai.DoMethod(id, "Cut")
```

FUNCTION
Cut currently selected text and put it in the clipboard.

INPUTS

`id`	id of the text editor object

49.10 Texteditor.GetLineLength

NAME
Texteditor.GetLineLength – get number of characters in line (V2.0)

SYNOPSIS
```
len = moai.DoMethod(id, "GetLineLength", line)
```

FUNCTION
This method returns the number of characters in the line specified by `line`. Trailing characters like carriage return or linefeed will not be included in the count. Line indices are counted from 0.

INPUTS

`id`	id of the text editor object
`line`	index of line to query (starting from 0)

RESULTS

`len`	number of characters in line

49.11 Texteditor.GetPosition

NAME
Texteditor.GetPosition – get index position from column and row (V2.0)

SYNOPSIS
```
pos = moai.DoMethod(id, "GetPosition", x, y)
```

FUNCTION
This method converts the column and row position specified by `x` and `y` into an index position. All values start at index 0. For invalid column and row positions, -1 is returned.

INPUTS

`id`	id of the text editor object
`x`	column position

`y` row position

RESULTS

`pos` index of the specified column and row position

49.12 Texteditor.GetSelection

NAME

Texteditor.GetSelection – get text selection

SYNOPSIS

```
start, end = moai.DoMethod(id, "GetSelection")
```

FUNCTION

This method returns the start and stop positions of the currently selected text. The first character of the text is at position 0. If there is no marked area, -1 is returned for both values.

INPUTS

`id` id of the text editor object

RESULTS

`start` start offset of marked text

`end` end offset of marked text

49.13 Texteditor.GetText

NAME

Texteditor.GetText – export portion of the current text

SYNOPSIS

```
t$ = moai.DoMethod(id, "GetText", start, end)
```

FUNCTION

This method exports the portion of the current text between `start` and `end` and returns it. Positions are counted from 0.

INPUTS

`id` id of the text editor object

`start` start position of desired block

`end` end position of desired block

RESULTS

`t$` text at the specified block coordinates

49.14 Texteditor.GetXY

NAME
Texteditor.GetXY – convert index position into column and row

SYNOPSIS
```
x, y = moai.DoMethod(id, "GetXY", pos)
```

FUNCTION
This method converts the specified index position into its column and row constituents. Both column and row counters start at index 0. For invalid indices, -1 is returned.

INPUTS

`id`	id of the text editor object
`pos`	index position to convert

RESULTS

`x`	column offset of the specified index
`y`	row offset of the specified index

49.15 Texteditor.HasChanged

NAME
Texteditor.HasChanged – learn about content change

FUNCTION
This attribute is set when the user changes the contents of the text. You can set up a notification on this attribute to learn about these changes.

Note that even though `Texteditor.HasChanged` will not trigger when setting `Texteditor.Text` or calling `Texteditor.Clear` those two will still lead `Texteditor.HasChanged` to be set to `False`.

TYPE
Boolean

APPLICABILITY
SGN

49.16 Texteditor.Hint

NAME
Texteditor.Hint – set/get text hint (V2.0)

FUNCTION
Set and get a texteditor widget's hint. The hint will be shown whenever there is no text in the widget. It can be used to give the user a hint concerning what he is expected to enter in the widget.

TYPE
String

APPLICABILITY
ISG

49.17 Texteditor.Insert

NAME
Texteditor.Insert – insert text

SYNOPSIS
```
moai.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)

49.18 Texteditor.Italic

NAME
Texteditor.Italic – 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.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

TYPE
Boolean

APPLICABILITY
SGN

49.19 Texteditor.Mark

NAME
Texteditor.Mark – mark text

SYNOPSIS
```
moai.DoMethod(id, "Mark", start, end)
```

FUNCTION
This method will mark the text in the area delimited by `start` and `end`. The first character starts at position 0.

INPUTS

`id`	id of the text editor object
`start`	start position
`end`	end position

49.20 Texteditor.MarkAll

NAME
Texteditor.MarkAll – mark all text

SYNOPSIS
```
moai.DoMethod(id, "MarkAll")
```

FUNCTION
Marks all text in the widget.

INPUTS

`id`	id of the text editor object

49.21 Texteditor.MarkNone

NAME
Texteditor.MarkNone – clear text selection

SYNOPSIS
```
moai.DoMethod(id, "MarkNone")
```

FUNCTION
Clears text selection.

INPUTS

`id`	id of the text editor object

49.22 Texteditor.NoWrap

NAME
Texteditor.NoWrap – disable wordwrapping

FUNCTION
By default, texteditor widgets will automatically use wordwrapping when words run beyond the available widget space. Set this tag to `True` if you don't want this. In that case, the texteditor widget will use a horizontal scrollbar instead of wrapping words to the next line.

TYPE
Boolean

APPLICABILITY
I

49.23 Texteditor.Paste

NAME
Texteditor.Paste – paste text from clipboard

SYNOPSIS
```
moai.DoMethod(id, "Paste")
```

FUNCTION
Pastes text from clipboard into the text editor widget.

INPUTS

`id`	id of the text editor object

49.24 Texteditor.ReadOnly

NAME
Texteditor.ReadOnly – put text editor in read-only mode

FUNCTION
Set this tag to `True` to put the widget into read-only mode.

This is probably of not much use since there is Textview class for displaying non-editable text. See Section 51.1 [Textview class], page 255, for details.

TYPE
Boolean

APPLICABILITY
I

49.25 Texteditor.Redo

NAME
Texteditor.Redo – redo last operation

SYNOPSIS
```
moai.DoMethod(id, "Redo")
```

FUNCTION
Redo last operation of text editor widget.

INPUTS

`id`	id of the text editor object

49.26 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 disable your redo button when there is nothing to redo.

TYPE
Boolean

APPLICABILITY
GN

49.27 Texteditor.ScrollToLine

NAME
Texteditor.ScrollToLine – scroll line into view (V2.0)

SYNOPSIS
```
moai.DoMethod(id, "ScrollToLine", line)
```

FUNCTION
Scroll the line specified by `line` into view. Line indices start at 0.

INPUTS

`id`	id of the text editor object
`line`	line to be scrolled into view (starting from 0)

49.28 Texteditor.SetBold

NAME

Texteditor.SetBold – toggle bold style of text block

SYNOPSIS

```
moai.DoMethod(id, "SetBold", start, end, flag)
```

FUNCTION

This method toggles bold style on a text block that is defined by `start` and `end` coordinates. 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.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

INPUTS

`id`	id of the text editor object
`start`	start position (starting from 0)
`end`	end position
`flag`	boolean flag that indicates toggle state

49.29 Texteditor.SetColor

NAME

Texteditor.SetColor – change color of text block

SYNOPSIS

```
moai.DoMethod(id, "SetColor", start, end, color)
```

FUNCTION

This method changes the color of a text block that is defined by `start` and `end` coordinates. The color has to be passed as a 24-bit RGB value.

To change the color that should be used for newly inserted text, use the `Texteditor.Color` attribute instead.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

INPUTS

`id`	id of the text editor object
`start`	start position (starting from 0)
`end`	end position
`color`	desired color for text

49.30 Texteditor.SetItalic

NAME

Texteditor.SetItalic – toggle italic style of text block

SYNOPSIS

```
moai.DoMethod(id, "SetItalic", start, end, flag)
```

FUNCTION

This method toggles italic style on a text block that is defined by `start` and `end` coordinates. 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.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

INPUTS

`id`	id of the text editor object
`start`	start position (starting from 0)
`end`	end position
`flag`	boolean flag that indicates toggle state

49.31 Texteditor.SetUnderline

NAME

Texteditor.SetUnderline – toggle underline style of text block

SYNOPSIS

```
moai.DoMethod(id, "SetUnderline", start, end, flag)
```

FUNCTION

This method toggles underline style on a text block that is defined by `start` and `stop` coordinates. 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.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

INPUTS

`id`	id of the text editor object
`start`	start position (starting from 0)
`end`	end position
`flag`	boolean flag that indicates toggle state

49.32 Texteditor.Styled

NAME
Texteditor.Styled – enable text formatting

FUNCTION
Set this attribute to `True` to enable text formatting for this widget. If set to `True`, you can use special text formatting codes in the text you pass to this widget. See Section 3.14 [Text formatting codes], page 22, for details.

TYPE
Boolean

APPLICABILITY
I

49.33 Texteditor.Text

NAME
Texteditor.Text – 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 in case `Texteditor.Styled` has been set to `True`. See Section 3.14 [Text formatting codes], page 22, for details.

Note that even though setting this attribute will not trigger a `Texteditor.HasChanged` notification it will set `Texteditor.HasChanged` to `False`.

TYPE
String

APPLICABILITY
SG

49.34 Texteditor.Underline

NAME
Texteditor.Underline – 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.

Note that this feature requires `Texteditor.Styled` to be set to `True`.

TYPE
Boolean

APPLICABILITY
SGN

49.35 Texteditor.Undo

NAME
Texteditor.Undo – undo last operation

SYNOPSIS
```
moai.DoMethod(id, "Undo")
```

FUNCTION
Undo last operation of text editor widget.

INPUTS

`id`	id of the text editor object

49.36 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 disable your undo button when there is nothing to undo.

TYPE
Boolean

APPLICABILITY
GN

50 Textentry class

50.1 Overview

Textentry class derives from Area class and creates single-line text entry widgets which can be used to enter a text string.

By default, text entry widgets do not have any label next to them. If you want to have a label next to your text entry widget, you need to put it into a `<hgroup>` and then use Label class to put a label next to it.

The XML tag's content is used as the initial contents of the text entry widget.

Here is an XML example of a text entry widget:

```
<textentry id="mytextentry"/>
```

Note that Textentry class only supports single line widgets. If you need a multi-line text entry widget, you have to use Texteditor class instead. See Section 49.1 [Texteditor class], page 233, for details.

50.2 Textentry.Accept

NAME
Textentry.Accept – set characters accepted by text entry widget

FUNCTION
Set this to a string containing characters that are accepted by the widget. Useful for example if you only want to allow the entering of numbers. In that case you would set `Textentry.Accept` to `"0123456789"`.

You can also use `Textentry.Reject` to selectively reject characters.

TYPE
String

APPLICABILITY
I

50.3 Textentry.Acknowledge

NAME
Textentry.Acknowledge – get notified when the user hits RETURN

FUNCTION
Whenever the user hits return this attribute will be set to `True`. You can listen to this notification and take the appropriate action.

Pressing the TAB key or clicking the mouse to deactivate the widget will not trigger `Textentry.Acknowledge`.

TYPE
Boolean

APPLICABILITY
N

50.4 Textentry.AdvanceOnCR

NAME
Textentry.AdvanceOnCR – activate next object when user hits RETURN

FUNCTION
If you set this to `True`, pressing RETURN will behave like pressing the TAB key, i.e. it will give the focus to the next widget in the window.

TYPE
Boolean

APPLICABILITY
I

50.5 Textentry.Copy

NAME
Textentry.Copy – copy marked text

SYNOPSIS
```
moai.DoMethod(id, "Copy")
```

FUNCTION
Copy currently selected text to clipboard.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS
`id` id of the textentry object

50.6 Textentry.CursorPos

NAME
Textentry.CursorPos – set/get cursor position

FUNCTION
Sets or gets the current cursor position.

On AmigaOS and compatibles this attributes requires MUI 4.0 or better.

TYPE
Number

APPLICABILITY
SG

50.7 Textentry.Cut

NAME
Textentry.Cut – cut marked text

SYNOPSIS
```
moai.DoMethod(id, "Cut")
```

FUNCTION
Cut currently selected text and put it in the clipboard.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS

`id`	id of the textentry object

50.8 Textentry.GetSelection

NAME
Textentry.GetSelection – get text selection

SYNOPSIS
```
start, end = moai.DoMethod(id, "GetSelection")
```

FUNCTION
Returns the range of the currently selected text. Offsets start at 0 which indicates the first character. If no text is selected, -1 is returned for both values.

INPUTS

`id`	id of the textentry object

RESULTS

`start`	start offset
`end`	end offset

50.9 Textentry.Hint

NAME
Textentry.Hint – set/get text hint (V2.0)

FUNCTION
Set and get a textentry widget's hint. The hint will be shown whenever there is no text in the widget. It can be used to give the user a hint concerning what he is expected to enter in the widget.

TYPE
String

APPLICABILITY
ISG

50.10 Textentry.Insert

NAME

Textentry.Insert – insert text

SYNOPSIS

```
moai.DoMethod(id, "Insert", t$)
```

FUNCTION

This will insert the given text `t$` at the current cursor position.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS

`id`	id of the textentry object
`t$`	text to insert

50.11 Textentry.Mark

NAME

Textentry.Mark – mark text

SYNOPSIS

```
moai.DoMethod(id, "Mark", start, end)
```

FUNCTION

Marks the text starting at the first position up to the character at the last position. Positions start at 0 which is the first character.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS

`id`	id of the textentry object
`start`	position of first character to mark
`end`	position of last character to mark

50.12 Textentry.MarkAll

NAME

Textentry.MarkAll – mark all the text

SYNOPSIS

```
moai.DoMethod(id, "MarkAll")
```

FUNCTION

Marks the all the text in the widget.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS

`id`	id of the textentry object

50.13 Textentry.MarkNone

NAME
Textentry.MarkNone – clear selection

SYNOPSIS
```
moai.DoMethod(id, "MarkNone")
```

FUNCTION
Removes any text selection. The text will be completely unmarked when this method returns.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS
`id` id of the textentry object

50.14 Textentry.MaxLen

NAME
Textentry.MaxLen – set maximum entry length

FUNCTION
Sets the maximum number of characters that can be entered.

TYPE
Number

APPLICABILITY
IG

50.15 Textentry.Password

NAME
Textentry.Password – put widget in password mode

FUNCTION
Set this attribute to hide the user input. Useful when entering passwords.

TYPE
Boolean

APPLICABILITY
I

50.16 Textentry.Paste

NAME
Textentry.Paste – paste text from clipboard

SYNOPSIS

```
moai.DoMethod(id, "Paste")
```

FUNCTION

Pastes text from clipboard into the text entry widget.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS

`id` id of the textentry object

50.17 Textentry.ReadOnly

NAME

Textentry.ReadOnly – disable text editing (V2.0)

FUNCTION

Set this tag to `True` to put the widget into read-only mode.

This is probably of not much use since there is Text class for displaying non-editable text. See Section 48.1 [Text class], page 231, for details.

TYPE

Boolean

APPLICABILITY

I

50.18 Textentry.Redo

NAME

Textentry.Redo – redo last operation

SYNOPSIS

```
moai.DoMethod(id, "Redo")
```

FUNCTION

Redo last operation of text entry widget.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS

`id` id of the textentry object

50.19 Textentry.Reject

NAME

Textentry.Reject – set characters rejected by text entry widget

FUNCTION
Set this to a string containing characters that should be rejected by the widget. For example, if you do not want to allow entering numbers you would set `Textentry.Reject` to `"0123456789"`.

You can also use `Textentry.Accept` to selectively accept characters.

TYPE
String

APPLICABILITY
I

50.20 Textentry.Text

NAME
Textentry.Text – set/get text entry widget contents

FUNCTION
Get and set a text entry widget's contents.

You can also set up a notification on this attribute to get notified whenever the widget's contents change. Note that this will lead to an event being emitted for every keystroke.

TYPE
String

APPLICABILITY
SGN

50.21 Textentry.Undo

NAME
Textentry.Undo – undo last operation

SYNOPSIS
```
moai.DoMethod(id, "Undo")
```

FUNCTION
Undo last operation of text entry widget.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS
`id` id of the textentry object

51 Textview class

51.1 Overview

Textview class derives from Area class and can be used to show larger amounts of text in a widget with support for wordwrapping and text formatting. The XML tag's content is used as the initial contents of the text view widget.

Here is an example of how to use the `<textview>` XML tag:

```
<textview>Hello World</textview>
```

Here is the same example in bold:

```
<textview styled="true">\33bHello World</textview>
```

The string you specify here can use text formatting codes if `Textview.Styled` has been set to `True`. See Section 3.14 [Text formatting codes], page 22, for details.

51.2 Textview.Align

NAME

Textview.Align – set/get text alignment

FUNCTION

Set or get the desired text alignment for multiline text. This can be one of the following values:

`Left`	Left alignment.
`Right`	Right alignment.
`Center`	Centered alignment.

TYPE

String (see above for possible values)

APPLICABILITY

ISG

51.3 Textview.Append

NAME

Textview.Append – append text to end of textview (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "Append", s$)
```

FUNCTION

Append the text in the string specified by `s$` to the end of the textview.

On AmigaOS and compatibles this method requires MUI 4.0 or better.

INPUTS

`id`	id of the textview object
`s$`	text to append

51.4 Textview.Styled

NAME
Textview.Styled – enable text formatting

FUNCTION
Set this attribute to `True` to enable text formatting for this widget. If set to `True`, you can use special text formatting codes in the text you pass to this widget. See Section 3.14 [Text formatting codes], page 22, for details.

TYPE
Boolean

APPLICABILITY
I

51.5 Textview.Text

NAME
Textview.Text – set/get contents of textview object

FUNCTION
Set or get the text to be shown by the textview widget. The text can contain linefeeds and is automatically wordwrapped once it reaches the widget's boundaries. Multiline text will be aligned according to the alignment set using `Textview.Align`.

The string you specify here can use text formatting codes if `Textview.Styled` has been set to `True`. See Section 3.14 [Text formatting codes], page 22, for details.

TYPE
String

APPLICABILITY
SG

52 Toolbar class

52.1 Overview

Toolbar class derives from Area class and creates a bar of image buttons which is typically placed at the top of a window. The buttons can be shown in a variety of different view modes: either as images, images and text, or just text. Usually toolbar buttons are shown as images only.

When declaring a toolbar widget 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 six buttons:

```
<toolbar>
   <button icon="1">Open</button>
   <button icon="2">Save</button>
   <button/>
   <button icon="3">Cut</button>
   <button icon="4">Copy</button>
   <button icon="5">Paste</button>
   <button/>
   <button icon="6">Help</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. Note the empty `<button/>` declarations: These will create spacer items to visually separate a group of buttons that belong together from the rest of the buttons. Toolbar buttons can use many more options like special images for selected and disabled states, tooltips, and more. See Section 53.1 [Toolbarbutton class], page 259, for details.

There can be only one toolbar per window and it always has to be the first child of the window's root group. RapaGUI won't accept toolbars at random positions in the GUI layout. Toolbars must always be the first child of the window's root group. Thus, they are placed either at the top (horizontal toolbars) or at the left of the window (vertical toolbars).

Also note that it is not possible to create stand-alone instances of this class using `moai.CreateObject()`. Toolbars always have to be created within the context of a window. Thus, if you want to create toolbars using `moai.CreateObject()`, you always have to create a complete window and embed the toolbar in this window declaration.

AmigaOS users please also take note that this class requires the `TheBar.mcc` extension to be installed.

52.2 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 `True`.

TYPE
Boolean

APPLICABILITY
I

52.3 Toolbar.ViewMode

NAME
Toolbar.ViewMode – set view mode of toolbar

FUNCTION
Sets the view mode of the toolbar. The following modes are supported:

`TextGfx`	Toolbar buttons appear as text and images.
`Gfx`	Toolbar buttons appear as images only. This is the default setting.
`Text`	Toolbar buttons appear as text only.

TYPE
String (see above for possible values)

APPLICABILITY
I

53 Toolbarbutton class

53.1 Overview

Toolbarbutton class is a subclass of toolbar class. It cannot be used on its own but must always be encapsulated inside a toolbar class definition. Toolbarbutton 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 52.1 [Toolbar class], page 257, for an example.

53.2 Toolbarbutton.Disabled

NAME
Toolbarbutton.Disabled – enable/disable button

FUNCTION
Enable/disable a toolbar button.

TYPE
Boolean

APPLICABILITY
ISG

53.3 Toolbarbutton.Help

NAME
Toolbarbutton.Help – set help text for toolbar button

FUNCTION
Set the toolbar button's help text. This text is automatically shown in the status bar of the window that hosts the toolbar whenever the mouse is moved over the toolbar button. This is quite convenient for the user because it explains the function of the individual toolbar buttons in an elegant way. See Section 46.1 [Statusbar class], page 227, for details.

Note that this is not the same as `Toolbarbutton.Tooltip`. Tooltips are shown as little windows right next to the toolbar button if the mouse has been hovering over it for some time. The help text, on the other hand, is shown immediately in the window's status bar as soon as the mouse cursor is over a toolbar button.

TYPE
String

APPLICABILITY
I

53.4 Toolbarbutton.Icon

NAME
Toolbarbutton.Icon – set toolbar button image

FUNCTION
Set this attribute to the identifier of a Hollywood brush or icon to add an image to your toolbar button. Whether this attribute expects a Hollywood brush or icon, depends on what you specify in the `Toolbarbutton.IconType` attribute. By default, `Toolbarbutton.Icon` expects a Hollywood brush.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

TYPE
Number

APPLICABILITY
I

53.5 Toolbarbutton.IconScale

NAME
Toolbarbutton.IconScale – configure automatic image scaling (V2.0)

FUNCTION
If `Toolbarbutton.Icon` has been set to a raster brush and RapaGUI is running on a high-DPI display, RapaGUI will automatically scale the brush's raster graphics to fit to the current monitor's DPI setting. If you don't want that, set this tag to `False`.

Alternatively, you can also globally disable automatic image scaling by setting the `ScaleGUI` tag to `False` when calling `@REQUIRE` on RapaGUI. See Section 3.4 [Initializing RapaGUI], page 11, for details.

Please also read the chapter about high-DPI support in RapaGUI to learn more about supporting high-DPI displays. See Section 3.12 [High-DPI support], page 20, for details.

TYPE
Boolean

APPLICABILITY
I

53.6 Toolbarbutton.IconType

NAME
Toolbarbutton.IconType – set icon type to use (V2.0)

FUNCTION

This attribute allows you to set the type of the Hollywood image object passed in the `Toolbarbutton.Icon` attribute. By default, `Toolbarbutton.Icon` expects a Hollywood brush. By setting `Toolbarbutton.IconType`, however, you can make it use a different Hollywood image type.

The following image types are currently available:

`Brush` Use a Hollywood brush. This is the default type. You can use either raster or vector brushes. Vector brushes have the advantage that they can be scaled to any resolution without losses in quality. This is very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

`Icon` Use a Hollywood icon. This image type has the advantage that it can contain several subimages of different sizes. This makes it possible to provide images in different resolutions which can be very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

Note that you can globally change the default of all `IconType` attributes to Hollywood icons by setting the `Application.UseIcons` tag. See Section 9.13 [Application.UseIcons], page 65, for details.

TYPE

String (see above for possible values)

APPLICABILITY

I

53.7 Toolbarbutton.Pressed

NAME

Toolbarbutton.Pressed – learn when a button is pressed

FUNCTION

This attribute is triggered when the user presses the button. RapaGUI automatically listens to this attribute for all toolbar buttons so you do not need to explicitly request a notification using the MOAI.Notify attribute.

TYPE

Boolean

APPLICABILITY

N

53.8 Toolbarbutton.Selected

NAME

Toolbarbutton.Selected – toggle selection state

FUNCTION
Use this to toggle the selection state of a toggle or radio button or listen to the current state of a toggle or radio button.

RapaGUI automatically listens to this attribute for all toolbar buttons so you do not need to explicitly request a notification using the MOAI.Notify attribute.

TYPE
Boolean

APPLICABILITY
ISGN

53.9 Toolbarbutton.Tooltip

NAME
Toolbarbutton.Tooltip – set tooltip for toolbar button

FUNCTION
Sets the toolbar button's tooltip.

Note that this is not the same as `Toolbarbutton.Help`. Tooltips are shown as little windows right next to the toolbar button if the mouse has been hovering over it for some time. The help text, on the other hand, is shown immediately in the window's status bar as soon as the mouse cursor is over a toolbar button.

TYPE
String

APPLICABILITY
I

53.10 Toolbarbutton.Type

NAME
Toolbarbutton.Type – set type of button to create

FUNCTION
This defines the type of toolbar button you would like to have. The following types are currently possible:

`Normal`	Normal toolbar button.
`Toggle`	Toolbar button that can be toggled between two states.
`Radio`	Toolbar button that forms a group with its neighbouring buttons. Only one member of the group can be selected at a time.

TYPE
String (see above for possible values)

APPLICABILITY
I

54 Treeview class

54.1 Overview

Treeview class derives from Area class and visualizes complex data structures using a tree model. The data is presented as a hierarchy in which every item can contain an infinite number of sub-items. Items that contain sub-items are called nodes whereas items with no sub-items are called leaves. RapaGUI's treeview class is very powerful and supports multi-column trees, checkboxes, editable nodes and leaves, and icons for the individual treeview items.

When creating a treeview in XML code, you always have to add at least one column to it. This is done by using Treeviewcolumn class. Here is an example of a minimal treeview declaration with just a single column:

```
<treeview>
   <column/>
</treeview>
```

It is also possible to add some entries to the treeview right at declaration time. This can be done by using the `<node>` and `<leaf>` tags which refer to Treeviewnode class and Treeviewleaf class, respectively. A treeview node always contains just a single entry, even for multi-column treeviews. Treeview leaves, however, have to declare as many items as there are columns in the treeview. Thus, you have to use one `<item>` tag per column inside the `<leaf>` to create these individual items. Here is an example declaration:

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

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

As you can see, we have created a single-column treeview with the XML code above. Thus, we only have to use `<item>` once per `<leaf>` declaration. For multi-column trees you'd have to use as many `<item>` tags as there are columns in your treeview. See Section 56.1 [Treeviewleaf class], page 283, for details. See Section 57.1 [Treeviewleafitem class], page 289, for details.

In the example we have also made use of the `Treeviewnode.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 58.1 [Treeviewnode class], page 291, for details. For example, to refer to an item (node or leaf) in a treeview object, you have to assign a unique ID string to it. You can then refer to this item by simply using this ID string. IDs are either assigned during object creation in XML code or when calling `Treeview.InsertNode` or `Treeview.InsertLeaf`.

In the example above we do not define any attributes for the treeview column. It is, however, possible to customize the appearance of treeview columns. For example, you can use the `Treeviewcolumn.Title` attribute to add a title bar to each of your columns. Furthermore, you can also add checkboxes to your columns and allow the editing of column items. See Section 55.1 [Treeviewcolumn class], page 277, for details.

Note that RapaGUI might use two different kinds of widgets for this class: In case you are creating a treeview that doesn't use any advanced functionality (such as multiple columns, checkboxes, editable items, decorations) RapaGUI might create a more basic treeview widget for you because some operating systems offer two different treeview-based widgets. RapaGUI will use the light widget in case your treeview doesn't use any of the advanced features because it's faster. If you don't want that, you can force RapaGUI to always give you a full-blown treeview by setting the `Treeview.ForceMode` attribute to the according tag. See Section 54.12 [Treeview.ForceMode], page 268, for details.

54.2 Treeview.AbortEditing

NAME

Treeview.AbortEditing – get notified when user cancels editing (V1.1)

FUNCTION

When setting up a notification on this attribute, RapaGUI will run your event callback whenever the user cancels an editing operation on an item, for example by pressing the escape key or clicking outside the item edit widget.

Note that you have to set `Treeviewcolumn.Editable` to `True` if leaves in the respective column shall be editable. To make nodes editable you have to set `Treeview.EditableNodes` to `True`.

Your event handler will be called with the following extra arguments:

`Item:` ID of the node or leaf that the user was editing when he cancelled the operation.

`Column:` Column index of the item the user was editing when he cancelled the operation. For nodes this is always 0.

See Section 3.7 [Notifications], page 13, for details.

TYPE
Boolean

APPLICABILITY
N

54.3 Treeview.Active

NAME
Treeview.Active – set/get active tree item

FUNCTION
Set this attribute to activate the specified tree item. You have to pass the id of the entry to be made active to this attribute. IDs are either assigned during object creation in XML code or when calling `Treeview.InsertNode` or `Treeview.InsertLeaf`.

To deselect the active entry, just pass the special value `Off`. When getting this attribute, it returns the id of the active tree item and `Off` in case there is no active entry.

You can also set up a notification on `Treeview.Active` to get informed whenever the selection changes. The active tree item's id will be returned as the `TriggerValue` then.

TYPE
String

APPLICABILITY
SGN

54.4 Treeview.Alternate

NAME
Treeview.Alternate – use alternating row colors

FUNCTION
Set this attribute to `True` to make the treeview appear with alternating row colors.

Note that on AmigaOS and compatibles, this feature requires at least MUI 5.0.

TYPE
Boolean

APPLICABILITY
I

54.5 Treeview.Clear

NAME
Treeview.Clear – clear treeview (V2.0)

SYNOPSIS
```
moai.DoMethod(id, "Clear")
```

FUNCTION
Remove all entries from treeview.

INPUTS

`id`	id of the treeview object

54.6 Treeview.ClickColumn

NAME
Treeview.ClickColumn – learn about column clicks (V2.0)

FUNCTION
In multi-column treeviews, this attribute records the number of the column where the user last clicked.

TYPE
Number

APPLICABILITY
GN

54.7 Treeview.Close

NAME
Treeview.Close – close tree node

SYNOPSIS
```
moai.DoMethod(id, "Close", node$)
```

FUNCTION
Closes the specified treeview node and all its children.

You can pass the following special values for `node$`:

`Root`	The root node. Passing this special value will close all nodes.
`Active`	The active node.

INPUTS

`id`	id of the treeview object
`node$`	id of tree node to use or special value (see above)

54.8 Treeview.DoubleClick

NAME

Treeview.DoubleClick – get notified about double clicks on treeview items

FUNCTION

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

TYPE

Boolean

APPLICABILITY

N

54.9 Treeview.DropFile

NAME

Treeview.DropFile – learn about dropped files (V1.1)

FUNCTION

When setting up a notification on this attribute, RapaGUI will run your event callback whenever one or more files have been dropped onto the treeview widget. The `TriggerValue` field of the message table will be set to a table that contains a list of all files that have been dropped onto the widget.

Additionally, the message table will contain the following two extra fields:

`X:` The x-position where the user has dropped the file(s). This will be relative to the widget's left corner.

`Y:` The y-position where the user has dropped the file(s). This will be relative to the widget's top corner.

See Section 3.7 [Notifications], page 13, for details.

Note that `Treeview.DropFile` is only ever triggered if `Treeview.DropTarget` has been set to `True` first.

TYPE

Boolean

APPLICABILITY

N

54.10 Treeview.DropTarget

NAME

Treeview.DropTarget – configure drop target settings (V1.1)

FUNCTION

Set this to `True` if files can be dropped on this treeview widget. You can listen to the `Treeview.DropFile` attribute to learn when the user drops one or more files on the treeview widget.

TYPE
Boolean

APPLICABILITY
ISG

54.11 Treeview.EditableNodes

NAME
Treeview.EditableNodes – allow editing of treeview nodes

FUNCTION
Set this to `True` to allow user editing of treeview nodes. The user will then be able to edit all nodes in the tree by executing a slow double click, i.e. slowly pressing left mouse button two times in a row.

If you would only like to allow editing of some nodes in the tree, you need to set this attribute to `True` and you need to listen to the `Treeview.StartEditing` attribute. `Treeview.StartEditing` will then be triggered whenever the user attempts to edit a node and your callback can return `False` to forbid editing of certain items. See Section 54.19 [Treeview.StartEditing], page 274, for details.

To get notified whenever the value of a treeview node changes because the user has edited it, you have to listen to the `Treeview.ValueChange` attribute.

To manually start editing of a treeview node, call the `Treeviewnode.Edit` method.

Please note that this attribute only applies to treeview nodes. If you would like leaf labels to be editable as well, you need to set the `Treeviewcolumn.Editable` attribute to `True`.

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

TYPE
Boolean

APPLICABILITY
I

54.12 Treeview.ForceMode

NAME
Treeview.ForceMode – override default treeview mode

FUNCTION
RapaGUI can use two different widgets for Treeview class depending on your settings. For example, in case a single-column treeview with no headings is used, RapaGUI might use a different widget for reasons of efficiency in case the host OS provides such a widget. For example, on Windows RapaGUI will use the Treeview control instead of a fully featured Dataview in those cases. If you don't want that, set this attribute to the desired widget and RapaGUI will try to use it.

The following modes are currently recognized:

`Normal` Automatically selects the widget that fits best. This is the default.

`Treeview` Use a Treeview widget. Treeview widgets only support single column trees with no editable items and no checkboxes and no titles and no special decorations.

`Dataview` Use a Dataview widget. Supports everything but currently uses a generic implementation on Windows.

Note that on AmigaOS and compatibles this attribute doesn't have any effect since RapaGUI always uses the same widget on those platforms.

TYPE
String (see above for possible values)

APPLICABILITY
I

54.13 Treeview.GetEntry

NAME
Treeview.GetEntry – get information about tree entry

SYNOPSIS
```
found, table = moai.DoMethod(id, "GetEntry", item, position)
```

FUNCTION
Get information about a tree entry. The tree entry can be specified either as an absolute index or also in relation to a specified node or leaf. This allows you to traverse the entire tree. See below for an example.

`item` specifies the tree item to use as a reference point. This can be a node or a leaf. Note that in contrast to all other methods or attributes of Treeview class, `item` must not be a string identifier but a special value returned by this method in the `UID` table field (see below) or obtained by querying `Treeviewleaf.UID` or `Treeviewnode.UID` for an item. Alternatively, it can be one of the following special values:

`Root` Use the root node.

`Active` Use the active item.

`position` can either be a number or a special value. Passing a number is only supported when the specified `item` is a node. In that case, the number indicates the index of the child you want to get information about, starting from 0, i.e. passing 5 here would return information about the sixth child of the node passed in `item`. Alternatively, you can pass the following special values in `position`:

`Head` Return information about the first child of the node.

`Tail` Return information about the last child of the node.

`Active` Return information about the active item.

`Next` Return the next entry in the tree after `item`.

`Previous` Return the previous entry in the tree before `item`.

`Parent` Return information about the parent of `item`.

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

`Items` This is a table containing the items for all columns of this treeview entry. Note that if the entry is a node, this table will only contain one item because nodes cannot span multiple columns.

`Node` `True` if the found entry is a node, `False` if it is a leaf.

`ID` String object identifier of this tree item.

`UID` Internal ID of this tree item. 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 `Treeview.GetEntry` in the `node` argument See above for more information and below for an example. You can also obtain UIDs by getting the `Treeviewleaf.UID` or `Treeviewnode.UID` attribute.

INPUTS

`id` id of the treeview object

`item` special identifier returned by this method or a special value (see above)

`position` index of entry to get or a special value (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$, node, indent)
   Local found, t = moai.DoMethod(id$, "GetEntry", node, "Head")
   While found = True
      If indent > 0
         DebugPrint(RepeatStr(" ", indent) ..
             IIf(t.Node = True, "+", ""), Unpack(t.items))
      Else
         DebugPrint(IIf(t.Node = True, "+", ""), Unpack(t.items))
      EndIf
      If t.Node = True Then p_DumpListTree(id$, t.uid, indent + 4)
      found, t = moai.DoMethod(id$, "GetEntry", t.uid, "Next")
    Wend
EndFunction

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

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

54.14 Treeview.HRules

NAME
Treeview.HRules – draw horizontal rules between rows

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set this to `True` to enable horizontal rules between rows for the treeview.

TYPE
Boolean

APPLICABILITY
I

54.15 Treeview.InsertLeaf

NAME
Treeview.InsertLeaf – insert new tree leaf

SYNOPSIS
```
moai.DoMethod(id, "InsertLeaf", id$, node$, pred$, [icon1,] entry1$, ...)
```

FUNCTION
Inserts a new leaf into the treeview at the position which is defined by `node$` and `pred$`. A leaf is a treeview item that doesn't have any children. `id$` must be a unique string identifier that you want to use to refer to the newly inserted treeview leaf.

The entry data for the new leaf has to be passed in the last parameters. If the treeview has multiple columns, you need to pass individual entry data for all the columns in the treeview. The entry data consists of a text string and, if the column has the `Treeviewcolumn.Icon` attribute set, an icon for each column. The icon has to be passed before the text string and it has to be an identifier of a Hollywood brush/icon which should be used as the icon for the entry. If `Treeviewcolumn.Icon` isn't set, then you must omit the icon parameter and only pass text data for the treeview entry. If you've set `Treeviewcolumn.Icon` to `True` and you don't want to show an icon in this particular row and column, you can also pass the special value -1. In that case, RapaGUI won't show an icon even though `Treeviewcolumn.Icon` has been set to `True`. Please note that auto-generated IDs cannot be used. Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

In case a column is showing a checkbox, you have to pass `"On"`, `"True"`, or `"1"` to select the checkbox and any other text to deselect the checkbox.

In `node$` you have to pass the node whose list is used to insert the new leaf. This can be the string identifier of a node or one of the following special values:

`Root` The root node.

In `pred$` you have to specify the node or leaf which will become the predecessor of the leaf to insert, i.e. the new leaf will be inserted after the item specified in `pred$`. This can be the string identifier of a node or a leaf or one of the following special values:

`Head` Insert as the first node child.

`Tail` Insert as the last node child.

`Active` Insert after the active item. If there is no active entry, the item will be inserted as the last node child.

Note that on AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

INPUTS

`id` id of the treeview object

`id$` unique string identifier for new treeview leaf

`node$` id of node to insert into or special value (see above)

`pred$` id of the predecessor node or leaf or special value (see above)

`icon1` optional: icon for entry in the first column; this must only be passed if the column has the icon attribute set; note that this must be a numeric identifier and auto-generated IDs are not valid in this case

`entry1$` entry to insert into the first column

`...` more entries if treeview has multiple columns

54.16 Treeview.InsertNode

NAME

Treeview.InsertNode – insert new tree node

SYNOPSIS

```
moai.DoMethod(id, "InsertNode", id$, node$, pred$, entry$[, icon])
```

FUNCTION

Inserts a new node into the treeview at the position which is defined by `node$` and `pred$`. A node is a treeview item that can have children and that can be opened by the user. `id$` must be a unique string identifier that you want to use to refer to the newly inserted treeview node.

`entry$` specifies the desired label for the new node. Optionally, you can also pass the id of a Hollywood brush/icon in the optional `icon` argument if you want to have an icon shown next to the node label. Note that node icons are always supported. You don't have to set `Treeviewcolumn.Icon` to `True` in order to show icons next to node labels. `Treeviewcolumn.Icon` only applies to leaves, nodes always support icons. Please note

that auto-generated IDs cannot be used. Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

In `node$` you have to pass the node whose list is used to insert the new node. This can be the string identifier of a node or one of the following special values:

`Root` The root node.

In `pred$` you have to specify the node or leaf which will become the predecessor of the node to insert, i.e. the new node will be inserted after the item specified in `pred$`. This can be the string identifier of a node or a leaf or one of the following special values:

`Head` Insert as the first node child.

`Tail` Insert as the last node child.

`Active` Insert after the active item. If there is no active entry, the item will be inserted as the last node child.

Note that on AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

INPUTS

`id` id of the treeview object

`id$` unique string identifier for new treeview node

`node$` id of node to insert into or special value (see above)

`pred$` id of predecessor node or leaf or special value (see above)

`entry$` desired node label

`icon` optional: desired node icon (defaults to -1 which means no icon); note that this must be a numeric identifier and auto-generated IDs are not valid in this case

54.17 Treeview.Open

NAME

Treeview.Open – open tree node

SYNOPSIS

```
moai.DoMethod(id, "Open", node$[, all])
```

FUNCTION

Opens the specified treeview node. If the optional argument `all` is set to `True`, all children of the specified node will be opened as well.

You can pass the following special values for `node$`:

`Root` The root node. Since the root node is never visible, this will automatically open all children of the invisible root node.

`Active` The active node.

INPUTS

`id`	id of the treeview object
`node$`	id of tree node to use or special value (see above)
`all`	optional: `True` to open all node children too (defaults to `False`)

54.18 Treeview.Remove

NAME

Treeview.Remove – remove tree item

SYNOPSIS

```
moai.DoMethod(id, "Remove", item$)
```

FUNCTION

Removes an item from a treeview (either a leaf or a complete node). You have to pass the id of the entry to remove in `item$`. Alternatively, you can also pass one of the following special values in `item$`:

`Active`	Removes the active item.
`Root`	Removes all items in the treeview. This does the same as calling the `Treeview.Clear` method. (V2.0)

INPUTS

`id`	id of the treeview object
`item$`	id of the tree item to remove or special value (see above)

54.19 Treeview.StartEditing

NAME

Treeview.StartEditing – get notified about item editing

FUNCTION

When setting up a notification on this attribute, RapaGUI will run your event callback whenever the user wants to edit a treeview item by slowly double-clicking on it or whenever your script runs the `Treeviewleaf.Edit` or `Treeviewnode.Edit` method. Your callback can then permit or forbid the user's edit request. To forbid the request, your callback has to return `False`. To permit the request, it has to return `True`.

Note that you have to set `Treeviewcolumn.Editable` to `True` if leaves in the respective column shall be editable. To make nodes editable you have to set `Treeview.EditableNodes` to `True`.

Your event handler will be called with the following extra arguments:

`Item:`	ID of the node or leaf that the user wants to edit.
`Column:`	Column index of the item which the user wants to edit. For nodes this is always 0.

See Section 3.7 [Notifications], page 13, for details.

TYPE
Boolean

APPLICABILITY
N

54.20 Treeview.ValueChange

NAME
Treeview.ValueChange – get notified when treeview item value changes

FUNCTION
When setting up a notification on this attribute, RapaGUI will run your event callback whenever a treeview item's value has changed because the user has toggled the checkbox or has edited the item. Note that `Treeview.ValueChange` will not trigger if the item's value was changed using the `Treeviewleaf.SetItem` or `Treeviewleaf.SetState` methods. It also won't trigger if `Treeviewnode.Name` was used to change the item's label.

For items in checkbox columns, `TriggerValue` will be set to either `True` or `False`, reflecting the new checkbox state. For items in text columns, `TriggerValue` will contain the new item text.

Additionally, your event handler will be called with the following extra arguments:

`Item:` ID of the item (node or leaf) whose value has changed.

`Column:` Column index of the item whose value has changed. For nodes this will always be 0.

See Section 3.7 [Notifications], page 13, for details.

TYPE
Boolean or string (depending on column type)

APPLICABILITY
N

54.21 Treeview.VRules

NAME
Treeview.VRules – draw vertical rules between columns

FUNCTION
Set this to `True` to enable vertical rules between columns for the treeview.

TYPE
Boolean

APPLICABILITY
I

55 Treeviewcolumn class

55.1 Overview

Treeviewcolumn class is needed when creating treeviews. It allows you to specify different attributes for the columns of your treeviews.

Treeviewcolumn class must always be embedded inside a `<treeview>` declaration. Its XML tag is `<column>`. See Section 54.1 [Treeview class], page 263, for details.

Note that you cannot create instances of this class using `moai.CreateObject()`. Treeview column numbers are currently static, i.e. you cannot add or remove columns at runtime.

55.2 Treeviewcolumn.Align

NAME
Treeviewcolumn.Align – set/get column alignment

FUNCTION
Set or get the column alignment. This can be one of the following values:

`Left` Left alignment. This is the default.

`Right` Right alignment.

`Center` Centered alignment.

On non-AmigaOS systems this attribute is only supported for the dataview backend. When creating a treeview and `Treeviewcolumn.Align` is set to a value other than `Left`, RapaGUI will automatically switch to the dataview backend. If you don't specify this attribute at creation time but want to set it later using `moai.Set()`, you have to explicitly request a dataview widget by setting the `Treeview.ForceMode` attribute.

TYPE
String (see above for possible values)

APPLICABILITY
ISG

55.3 Treeviewcolumn.Checkbox

NAME
Treeviewcolumn.Checkbox – put column in checkbox mode

FUNCTION
Set this to `True` to mark this column as a checkbox column. Checkbox columns show checkboxes instead of text. Whenever an item's text in a checkbox column is set to `"On"`, `"True"`, or `"1"`, the checkbox will be selected. All other item texts will lead to an unselected checkbox.

You can modify the states of the checkboxes by using the `Treeviewleaf.SetState` method. Similarly, getting the state of a checkbox is possible via the `Treeviewleaf.GetState` method.

To get notified whenever the user toggles a checkbox state, you have to listen to the `Treeview.ValueChange` attribute.

Also note that `Treeviewcolumn.Checkbox` and `Treeviewcolumn.Editable` and `Treeviewcolumn.Icon` are mutually exclusive. You cannot create checkbox columns that are editable or show icons.

Checkboxes in the first treeview column are currently unsupported on Windows, macOS, and Linux.

TYPE
Boolean

APPLICABILITY
I

55.4 Treeviewcolumn.Editable

NAME
Treeviewcolumn.Editable – allow editing of column leaves

FUNCTION
Set this to `True` to allow user editing of the leaves in this column. The user will then be able to edit all leaves in this column by executing a slow double click, i.e. slowly pressing left mouse button two times in a row.

If you would only like to allow editing of some leaves in the column, you need to set this attribute to `True` and you need to listen to the `Treeview.StartEditing` attribute. `Treeview.StartEditing` will then be triggered whenever the user attempts to edit a leaf and your callback can return `False` to forbid editing of certain items. See Section 54.19 [Treeview.StartEditing], page 274, for details.

To get notified whenever the value of a treeview leaf changes because the user has edited it, you have to listen to the `Treeview.ValueChange` attribute.

To manually start editing of a treeview leaf, call the `Treeviewleaf.Edit` method.

Please note that this attribute only applies to treeview leaves. If you would like node labels to be editable as well, you need to set the `Treeview.EditableNodes` attribute to `True`.

Also, `Treeviewcolumn.Checkbox` and `Treeviewcolumn.Editable` are mutually exclusive. You cannot create editable checkbox columns.

Note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

TYPE
Boolean

APPLICABILITY
I

55.5 Treeviewcolumn.Hide

NAME
Treeviewcolumn.Hide – show/hide treeview column

FUNCTION
This attribute allows you to show or hide single treeview columns. Note that the columns will still be there, they'll just be invisible. Thus, you must not forget hidden columns when adding treeview leaves using `Treeview.InsertLeaf`.

On non-AmigaOS systems this attribute is only supported for the dataview backend. When creating a treeview and `Treeviewcolumn.Hide` is set to `True`, RapaGUI will automatically switch to the dataview backend. If you don't specify this attribute at creation time but want to set it later using `moai.Set()`, you have to explicitly request a dataview widget by setting the `Treeview.ForceMode` attribute.

TYPE
Boolean

APPLICABILITY
ISG

55.6 Treeviewcolumn.Icon

NAME
Treeviewcolumn.Icon – enable leaf icons for this column

FUNCTION
Set this to `True` if treeview leaves in this column use icons. In that case, you have to pass Hollywood brushes/icons to use as icons to the `Treeview.InsertLeaf` method.

Note that this attribute only applies to treeview leaves. Treeview nodes can use icons without setting any specific attribute. Icons for treeview nodes are always available, but for leaves you have to set this attribute to `True` first.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

On AmigaOS and compatibles icon support is only available with MUI 4.0 or better.

TYPE
Boolean

APPLICABILITY
I

55.7 Treeviewcolumn.IconScale

NAME

Treeviewcolumn.IconScale – configure automatic image scaling (V2.0)

FUNCTION

If `Treeviewcolumn.Icon` has been set to a raster brush and RapaGUI is running on a high-DPI display, RapaGUI will automatically scale the brush's raster graphics to fit to the current monitor's DPI setting. If you don't want that, set this tag to `False`.

Alternatively, you can also globally disable automatic image scaling by setting the `ScaleGUI` tag to `False` when calling `@REQUIRE` on RapaGUI. See Section 3.4 [Initializing RapaGUI], page 11, for details.

Please also read the chapter about high-DPI support in RapaGUI to learn more about supporting high-DPI displays. See Section 3.12 [High-DPI support], page 20, for details.

TYPE

Boolean

APPLICABILITY

I

55.8 Treeviewcolumn.IconType

NAME

Treeviewcolumn.IconType – set icon type to use (V2.0)

FUNCTION

This attribute allows you to set the type of the Hollywood image object passed in the `Treeviewcolumn.Icon` attribute. By default, `Treeviewcolumn.Icon` expects a Hollywood brush. By setting `Treeviewcolumn.IconType`, however, you can make it use a different Hollywood image type.

The following image types are currently available:

`Brush` Use a Hollywood brush. This is the default type. You can use either raster or vector brushes. Vector brushes have the advantage that they can be scaled to any resolution without losses in quality. This is very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

`Icon` Use a Hollywood icon. This image type has the advantage that it can contain several subimages of different sizes. This makes it possible to provide images in different resolutions which can be very useful when designing applications that should be compatible with high-DPI monitors. See Section 3.12 [High-DPI support], page 20, for details.

Note that you can globally change the default of all `IconType` attributes to Hollywood icons by setting the `Application.UseIcons` tag. See Section 9.13 [Application.UseIcons], page 65, for details.

TYPE

String (see above for possible values)

APPLICABILITY
I

55.9 Treeviewcolumn.Title

NAME
Treeviewcolumn.Title – set/get column title

FUNCTION
Set or get the title for the column. The title is always shown at the top of the treeview and doesn't go away when the treeview is scrolled.

TYPE
String

APPLICABILITY
ISG

55.10 Treeviewcolumn.Width

NAME
Treeviewcolumn.Width – set/get column width

FUNCTION
Set or get the column width in device-independent pixels. This defaults to -1 which means that the column should be made as large as its largest entry.

TYPE
Number

APPLICABILITY
ISG

56 Treeviewleaf class

56.1 Overview

Treeviewleaf class is needed when creating treeviews. It allows you to define the single leaves of the treeview and set different attributes for them. Each leaf of your treeview has to contain an entry for every column of your treeview. Thus, your `<leaf>` declarations must include as many `<item>` declarations as there are columns in your treeview. See Section 57.1 [Treeviewleafitem class], page 289, for details.

Treeviewleaf class must always be embedded inside a `<treeview>` declaration. See Section 54.1 [Treeview class], page 263, for details.

Note that you cannot create instances of this class using `moai.CreateObject()`. Instead, you have to use `Treeview.InsertLeaf` to create leaves at runtime.

56.2 Treeviewleaf.Edit

NAME

Treeviewleaf.Edit – prompt user to edit a leaf item

SYNOPSIS

```
moai.DoMethod(id, "Edit", column)
```

FUNCTION

This method can be used to programmatically initiate leaf item editing. Normally, item editing is started by the user by slowly double-clicking an item. This method provides an alternative to this user mechanism.

This will only work if `Treeviewcolumn.Editable` has been set to `True` for the respective treeview column.

When the user has finished editing, the `Treeview.ValueChange` attribute will be triggered.

Note that if you have installed a listener on the `Treeview.StartEditing` attribute, then this callback will be asked for permission first before editing is actually started.

To learn about editing operations getting cancelled, you can listen to the `Treeview.AbortEditing` attribute.

Also note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

INPUTS

`id`	id of the leaf
`column`	column index of the leaf item to edit

56.3 Treeviewleaf.GetDisabled

NAME
Treeviewleaf.GetDisabled – get checkbox disabled state

SYNOPSIS
```
state = moai.DoMethod(id, "GetDisabled", column)
```

FUNCTION
Returns the disabled state of the checkbox in the specified column. This is either `True` or `False`.

INPUTS

`id`	id of the leaf
`column`	column index of the checkbox

RESULTS

`state`	`True` if the checkbox is disabled, `False` otherwise

56.4 Treeviewleaf.GetIcon

NAME
Treeviewleaf.GetIcon – get leaf item icon

SYNOPSIS
```
br = moai.DoMethod(id, "GetIcon", column)
```

FUNCTION
Returns the Hollywood brush/icon id of the leaf icon used in the specified column. If the specified item doesn't have an icon, -1 is returned.

INPUTS

`id`	id of the leaf
`column`	column index of the item

RESULTS

`br`	id of a Hollywood brush/icon or -1 if there is no icon

56.5 Treeviewleaf.GetItem

NAME
Treeviewleaf.GetItem – get leaf item label

SYNOPSIS
```
t$ = moai.DoMethod(id, "GetItem", column)
```

FUNCTION
Returns the label text of the item in the specified leaf column.

INPUTS

`id`	id of the leaf
`column`	column index of the item

RESULTS

`t$`	label of the specified item

56.6 Treeviewleaf.GetState

NAME

Treeviewleaf.GetState – get checkbox toggle state

SYNOPSIS

```
state = moai.DoMethod(id, "GetState", column)
```

FUNCTION

Returns the toggle state of the checkbox in the specified column. This is either `True` if the checkbox is selected or `False` otherwise.

INPUTS

`id`	id of the leaf
`column`	column index of the checkbox

RESULTS

`state`	`True` if the checkbox is selected, `False` otherwise

56.7 Treeviewleaf.SetDisabled

NAME

Treeviewleaf.SetDisabled – set checkbox disabled state

SYNOPSIS

```
moai.DoMethod(id, "SetDisabled", column, state)
```

FUNCTION

Sets the disabled state of the checkbox in the specified column. Pass `True` to disable the checkbox or `False` to enable it.

INPUTS

`id`	id of the leaf
`column`	column index of the checkbox
`state`	`True` if the checkbox should be disabled, `False` otherwise

56.8 Treeviewleaf.SetIcon

NAME

Treeviewleaf.SetIcon – set leaf item icon

SYNOPSIS

```
moai.DoMethod(id, "SetIcon", column, br)
```

FUNCTION

Adds an icon to the specified leaf item. You have to pass the id of a Hollywood brush/icon which should be used as the icon in `br`. To remove an icon from a leaf item, pass -1 in `br`.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

INPUTS

`id`	id of the leaf
`column`	column index of the leaf item
`br`	identifier of a Hollywood brush/icon to use as an icon or -1 to remove the item icon

56.9 Treeviewleaf.SetItem

NAME

Treeviewleaf.SetItem – set leaf item label

SYNOPSIS

```
moai.DoMethod(id, "SetItem", column, l$)
```

FUNCTION

Changes the label text of the specified leaf item to l$.

INPUTS

`id`	id of the leaf
`column`	column index of the leaf item
`l$`	desired new label text for the leaf item

56.10 Treeviewleaf.SetState

NAME

Treeviewleaf.SetState – set checkbox toggle state

SYNOPSIS

```
moai.DoMethod(id, "SetState", column, state)
```

FUNCTION

Sets the toggle state of the checkbox in the specified column. Pass `True` to select the checkbox or `False` to unselect it.

INPUTS

`id`	id of the leaf
`column`	column index of the checkbox
`state`	`True` if the checkbox should be selected, `False` otherwise

56.11 Treeviewleaf.UID

NAME

Treeviewleaf.UID – get leaf UID

FUNCTION

Gets the UID of the specified tree leaf. This UID is needed when running the `Treeview.GetEntry` method.

TYPE

MOAI object

APPLICABILITY

G

57 Treeviewleafitem class

57.1 Overview

Treeviewleafitem class is needed when creating treeviews. It allows you to define the individual items for each column of a treeview leaf. You have to use as many `<item>` declarations as there are columns in your treeview. Since this class is a subclass of Treeviewleaf class, your `<item>` declarations have to be embedded inside the `<leaf>` tag. Leaf items can have a label and optionally an icon. See Section 54.1 [Treeview class], page 263, for an example.

Note that you cannot create instances of this class using `moai.CreateObject()`. Instead, you have to use `Treeview.InsertLeaf` to create leaf items at runtime.

57.2 Treeviewleafitem.Icon

NAME

Treeviewleafitem.Icon – set item image

FUNCTION

Set this attribute to the identifier of a Hollywood brush or icon to add an image to your treeview leaf item. Whether this attribute expects a Hollywood brush or icon, depends on what you specify in the `Treeviewcolumn.IconType` attribute. By default, `Treeviewleafitem.Icon` expects a Hollywood brush.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

TYPE

Number

APPLICABILITY

I

58 Treeviewnode class

58.1 Overview

Treeviewnode class is needed when creating treeviews. It allows you to define the single nodes of the treeview and set different attributes for them. Note that in contrast to treeview leaves, nodes only contain a single entry, even for multi-column treeviews.

Treeviewnode class must always be embedded inside a `<treeview>` declaration. See Section 54.1 [Treeview class], page 263, for details.

Note that you cannot create instances of this class using `moai.CreateObject()`. Instead, you have to use `Treeview.InsertNode` to create nodes at runtime.

58.2 Treeviewnode.Edit

NAME

Treeviewnode.Edit – prompt user to edit a node label

SYNOPSIS

```
moai.DoMethod(id, "Edit")
```

FUNCTION

This method can be used to programmatically initiate node label editing. Normally, node label editing is started by the user by slowly double-clicking an item. This method provides an alternative to this user mechanism.

This will only work if `Treeview.EditableNodes` has been set to `True` in the treeview.

When the user has finished editing, the `Treeview.ValueChange` attribute will be triggered.

Note that if you have installed a listener on the `Treeview.StartEditing` attribute, then this callback will be asked for permission first before editing is actually started.

To learn about editing operations getting cancelled, you can listen to the `Treeview.AbortEditing` attribute.

Also note that on AmigaOS and compatibles this feature is only available on MUI 4.0 or higher.

INPUTS

`id` id of the node

58.3 Treeviewnode.Icon

NAME

Treeviewnode.Icon – set node image

FUNCTION

Set this attribute to the identifier of a Hollywood brush or icon to add an image to your treeview node. Whether this attribute expects a Hollywood brush or icon, depends on what you specify in the `Treeviewcolumn.IconType` attribute. By default,

`Treeviewnode.Icon` expects a Hollywood brush. To remove the icon from a tree node, set this attribute to -1.

Note that RapaGUI might scale the image to fit to the current monitor's DPI setting. Please read the chapter on high-DPI support for more information. See Section 3.12 [High-DPI support], page 20, for details.

Please also read about RapaGUI's image cache to learn more about icon support in RapaGUI. See Section 3.20 [Image cache], page 30, for details.

TYPE
Number

APPLICABILITY
ISG

58.4 Treeviewnode.Name

NAME
Treeviewnode.Name – set/get node name

FUNCTION
Sets or gets the string that is used as the node name.

When creating a new node, this attribute is mandatory and must always be set.

TYPE
String

APPLICABILITY
ISG

58.5 Treeviewnode.UID

NAME
Treeviewnode.UID – get node UID

FUNCTION
Gets the UID of the specified tree node. This UID is needed when running the `Treeview.GetEntry` method.

TYPE
MOAI object

APPLICABILITY
G

59 VLine class

59.1 Overview

VLine class derives from Area class and creates a vertical divider line which can be used separate groups of widgets. There is also HLine class which creates horizontal divider lines. See Section 19.1 [HLine class], page 119, for details.

VLine class doesn't define any attributes.

60 VSpace class

60.1 Overview

VSpace class simply creates objects of a fixed device-independent pixel size. This is typically used to fine-tune the GUI layout. To create padding objects that are freely resizable, you can use Rectangle class instead. See Section 41.1 [Rectangle class], page 209, for details.

60.2 VSpace.Height

NAME
VSpace.Height – set vertical space

FUNCTION
Sets the desired vertical space for this object in device-independent pixels.

TYPE
Number

APPLICABILITY
I

61 VSplitter class

61.1 Overview

VSplitter class is a special variant of Group class. It creates a vertical group of two children with a sash between them that allows the user to individually adjust the sizes of the group's two children by dragging the sash.

Note that VSplitter groups must always contain exactly two children. Additionally, the children must also be sizable because VSplitter class allows its children to be resized. Here is an example XML excerpt for creating a vertical splitter layout with two textviews:

```
<vsplitter>
    <textview>One</textview>
    <textview>Two</textview>
</vsplitter>
```

VSplitter class is available since RapaGUI 2.0.

61.2 VSplitter.Border

NAME
VSplitter.Border – set sash border style (V2.0)

FUNCTION
Set this to `False` to make the sash use a different border style. Note that this is not supported on all platforms.

TYPE
Boolean

APPLICABILITY
I

61.3 VSplitter.Gravity

NAME
VSplitter.Gravity – set/get sash gravity (V2.0)

FUNCTION
Set or get the sash gravity. Gravity is a value between 0 and 100 which controls the position of the sash while resizing the splitter group. The gravity value tells splitter group how much the top child will grow while resizing. For example:

- 0: Only the bottom child is automatically resized.
- 50: Both children grow by equal size.
- 100: Only the top child grows.

The default sash gravity is 0.

TYPE
Number

APPLICABILITY
SG

61.4 VSplitter.MinPaneSize

NAME
VSplitter.MinPaneSize – set/get minimum pane size (V2.0)

FUNCTION
Set or get the minimum pane size. The default minimum pane size is 0, which means that either pane can be reduced to zero by dragging the sash, thus removing one of the panes. To prevent this behaviour (and veto out-of-range sash dragging), set a minimum size, for example 20.

This value is in device-independent pixels.

TYPE
Number

APPLICABILITY
ISG

61.5 VSplitter.Position

NAME
VSplitter.Position – set/get sash position (V2.0)

FUNCTION
Set or get the position of the sash dividing the group's children. This value is in device-independent pixels.

TYPE
Number

APPLICABILITY
SG

61.6 VSplitter.Split

NAME
VSplitter.Split – split the panes (V2.0)

SYNOPSIS
```
moai.DoMethod(id, "Split", pos)
```

FUNCTION
This splits the group into two panes. The `pos` argument specifies the position of the sash in device-independent pixels. If this value is positive, it specifies the size of the top pane. If it is negative, its absolute value gives the size of the bottom pane. Finally, specify 0 to choose the default position (half of the total window height).

INPUTS

`id`	id of the splitter object
`pos`	desired sash position

61.7 VSplitter.Unsplit

NAME

VSplitter.Unsplit – unsplit the panes (V2.0)

SYNOPSIS

```
moai.DoMethod(id, "Unsplit", idx)
```

FUNCTION

This unsplits the panes and hides the pane specified in `idx`. If `idx` is 0, the top pane will be hidden, if it is 1, the bottom pane will be hidden. To make both panes visible again, use the `VSplitter.Split` method.

INPUTS

`id`	id of the splitter object
`idx`	index of pane to remove

62 Window class

62.1 Overview

Window class creates top-level windows that can be filled with groups of widgets. These groups are children of the top-level windows. As such, groups are automatically relayouted whenever the size of their parent changes. This allows you to create flexible GUI layouts that automatically adapt to the available screen space. Additionally, you can also attach menubars and toolbars to a window.

Windows are children of Application class. When creating window objects dynamically, you first have to add them to the application object by calling `Application.AddWindow`.

If you need to open dialogs, you can do so by creating objects of Dialog class. Dialogs are special top-level windows which block the rest of the application until they are closed. Dialogs are typically used when user action is required to continue a task or to indicate that the application is currently busy. A dialog could show a progress bar then, for example. See Section 16.1 [Dialog class], page 95, for details.

Here is a minimal example of creating a window in XML:

```
<window title="Hello World!">
   <vgroup>
       <button id="btn">Hello World!</button>
   </vgroup>
</window>
```

Note that the root element of a window always needs to be a single group object, i.e. an instance of Group class. See Section 18.1 [Group class], page 105, for details. In our example, we use a `<vgroup>` as the root element. It is not allowed to have multiple elements at the window's root level. You must only use a single group object as the root element.

Windows that are declared in XML code that is parsed by `moai.CreateApp()` are opened automatically unless you explicitly request them to stay closed by setting `Window.Open` to `False`. Windows created by `moai.CreateObject()`, on the other hand, are not opened automatically. You first have to add them to the application's window list by calling `Application.AddWindow` and then set `Window.Open` to `True`.

62.2 Window.Accelerator

NAME

Window.Accelerator – set window's accelerator table

FUNCTION

Set the accelerator table for this window. See Section 7.1 [Accelerator class], page 57, for details.

TYPE

MOAI object

APPLICABILITY

I

62.3 Window.Activate

NAME
Window.Activate – change activation state of window

FUNCTION
Set this attribute to `True` to activate the window.

You can also set up a notification on this attribute to learn whenever the window is activated or deactivated.

TYPE
Boolean

APPLICABILITY
ISGN

62.4 Window.ActiveObject

NAME
Window.ActiveObject – set/get active widget

FUNCTION
Set or get the active widget. The active widget is the one that has the keyboard focus. Widgets can also be activated manually by using the TAB key.

TYPE
MOAI object

APPLICABILITY
SG

62.5 Window.Borderless

NAME
Window.Borderless – make window borderless (V2.0)

PLATFORMS
Amiga, Android

FUNCTION
Set this to `True` to create a window without a frame.

TYPE
Boolean

APPLICABILITY
I

62.6 Window.CloseGadget

NAME
Window.CloseGadget – configure window's close gadget

FUNCTION
Set this to `False` to create a window without a close gadget.

TYPE
Boolean

APPLICABILITY
I

62.7 Window.CloseRequest

NAME
Window.CloseRequest – handle close request of window

FUNCTION
If you set up a notification on this attribute, the window won't be closed automatically when the user presses the window's close gadget. Instead, your event handler will be called and you have to close the window manually by setting `Window.Open` to `False`. Or you may choose to keep the window open if there are things that have to be finished first.

A typical use case for this attribute is to ask the user whether he wants to save the current project before terminating the application.

If there is no notification on this attribute, windows will be closed automatically when the close gadget is pressed.

TYPE
Boolean

APPLICABILITY
N

62.8 Window.DefaultObject

NAME
Window.DefaultObject – set/get window's default widget

FUNCTION
The default widget is the one that is active when a window is opened. Typically, this could be an "OK" or another confirmation button so that the user can just hit RETURN to close the window.

TYPE
MOAI object

APPLICABILITY
ISG

62.9 Window.DragBar

NAME
Window.DragBar – configure window's drag bar

FUNCTION
Set this to `False` if you do not want your window to be draggable.

TYPE
Boolean

APPLICABILITY
I

62.10 Window.Height

NAME
Window.Height – set/get window height

FUNCTION
Set the window height in device-independent pixels. This can either be an absolute pixel value or one of the following special values:

`Default` Calculate height from the default sizes of all widgets.

`Screen:<1..100>`
Set height as a percentage of the host screen's total height.

Default for this tag is `Default`.

TYPE
Number or predefined macro

APPLICABILITY
IG

62.11 Window.HideFromTaskbar

NAME
Window.HideFromTaskbar – configure taskbar visibility

PLATFORMS
Windows, Linux

FUNCTION
Set this to `True` if your window shouldn't appear in the taskbar.

TYPE
Boolean

APPLICABILITY
I

62.12 Window.Left

NAME
Window.Left – set/get left edge of window

FUNCTION
Set the horizontal position of the window. You can pass an absolute value in device-independent pixels here or use one of the following macros:

`Centered` Center window in the visible area of screen.

`Moused` Open window under the mouse cursor.

Default for this tag is `Centered`.

TYPE
Number or predefined macro

APPLICABILITY
IG

62.13 Window.Margin

NAME
Window.Margin – set window margin

FUNCTION
Sets the window margin in device-independent pixels. The window margin is defined as the space between the window's frame and the root group.

TYPE
Number

APPLICABILITY
I

62.14 Window.MaximizeGadget

NAME
Window.MaximizeGadget – configure window's maximize gadget

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set this to `False` if you don't want to have a maximize gadget for your window. Defaults to `False` if `Window.Toolwindow` is set, otherwise to `True`.

TYPE
Boolean

APPLICABILITY
I

62.15 Window.Menubar

NAME
Window.Menubar – set window's menubar

FUNCTION
Set the menubar for this window. See Section 31.1 [Menubar class], page 175, for details.

TYPE
MOAI object

APPLICABILITY
I

62.16 Window.MinimizeGadget

NAME
Window.MinimizeGadget – configure window's minimize gadget

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set this to `False` if you don't want to have a minimize gadget for your window. Defaults to `False` if `Window.Toolwindow` is set, otherwise to `True`.

TYPE
Boolean

APPLICABILITY
I

62.17 Window.NoCyclerMenu

NAME
Window.NoCyclerMenu – do not add a window cycler menu (V2.0)

PLATFORMS
Android

FUNCTION
By default, RapaGUI on Android will allow you to switch between individual windows using the menu. If you don't want that, set this attribute to `True`. In that case, RapaGUI won't add a cycler menu to the application's menu. Note that the cycler menu will of course only be added when multiple windows are open.

TYPE
Boolean

APPLICABILITY
I

62.18 Window.Open

NAME
Window.Open – open/close a window

FUNCTION
Set this attribute to open and close your windows.

Note that windows that are declared in XML code that is parsed by `moai.CreateApp()` are opened automatically unless you explicitly request them to stay closed by setting `Window.Open` to `False`. Windows created by `moai.CreateObject()`, on the other hand, are not opened automatically. You first have to add them to the application's window list by calling `Application.AddWindow` and then set `Window.Open` to `True`.

TYPE
Boolean

APPLICABILITY
ISG

62.19 Window.Orientation

NAME
Window.Orientation – set/get window orientation (V2.0)

PLATFORMS
Android

FUNCTION
Set or get the current window orientation. This can be one of the following special values:

`Device` Use the current device orientation. This is the default.

`Landscape`
Use landscape orientation.

`Portrait` Use portrait orientation.

You can also set up a notification on this attribute to learn when the user rotates the device.

TYPE
String (see above for possible values)

APPLICABILITY
ISGN

62.20 Window.Parent

NAME
Window.Parent – set window parent

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set this to an identifier of a MOAI object that should become the parent of this window. If a window has a parent, the window will automatically be hidden when the parent is hidden and it will also be centered above the parent in case no explicit position has been specified.

Note that you should use Dialog class if you want the parent window to be blocked while the child is open. See Section 16.1 [Dialog class], page 95, for details.

TYPE
MOAI object

APPLICABILITY
I

62.21 Window.PubScreen

NAME
Window.PubScreen – set window's screen

PLATFORMS
AmigaOS and compatibles

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

62.22 Window.Remember

NAME
Window.Remember – remember window position and size (V2.0)

FUNCTION
Set this attribute to `True` to if this window should remember its position and size when the program is restarted (or when dialogs are destroyed and re-created).

Note that this is only possible for windows/dialogs that have an ID. The ID is used to memorize the window's position and size in order to restore it later so keep in mind that if you change the window's ID, its position and size settings will get lost (or they will be transferred to another window in case you re-use an ID previously used for a certain window for a new window)

Also note that this attribute defaults to `True` on Amiga systems because the standard MUI behaviour is to always remember positions. On all other systems it defaults to `False`.

TYPE
Boolean

APPLICABILITY
I

62.23 Window.ScreenTitle

NAME
Window.ScreenTitle – set/get screen title of window

PLATFORMS
AmigaOS and compatibles

FUNCTION
Set the text that is shown in the screen's title bar when the window is active.

TYPE
String

APPLICABILITY
ISG

62.24 Window.SingleMenu

NAME
Window.SingleMenu – use a single menu (V2.0)

PLATFORMS
Android

FUNCTION
If this attribute is set to `True`, only the first menu of the menu bar will be used and its items will be shown directly when pressing the options button in the action bar. All other menus that might be in the menu bar will be ignored.

TYPE
Boolean

APPLICABILITY
I

62.25 Window.SizeChange

NAME
Window.SizeChange – learn about size changes (V2.0)

FUNCTION
This attribute will trigger a notification whenever the user changes the window size. Your event callback will receive two additional parameters named `Width` and `Height` which will contain the new dimensions of the window.

TYPE
Boolean

APPLICABILITY
N

62.26 Window.SizeGadget

NAME
Window.SizeGadget – configure window's size gadget

FUNCTION
Set this to `False` if you don't want to have a size gadget for your window.

TYPE
Boolean

APPLICABILITY
I

62.27 Window.StayOnTop

NAME
Window.StayOnTop – open window which stays on top

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set this to `True` to make your window stay on top of the window Z-order.

TYPE
Boolean

APPLICABILITY
I

62.28 Window.Subtitle

NAME
Window.Subtitle – set/get window subtitle (V2.0)

PLATFORMS
Android

FUNCTION
Set the text that is shown as the action bar's subtitle.

TYPE
String

APPLICABILITY
ISG

62.29 Window.Toolwindow

NAME
Window.Toolwindow – designate window as a tool window

PLATFORMS
Windows, Linux, macOS

FUNCTION
Set this to `True` to make your window open in the tool window design. Tool windows usually have a smaller frame than normal top-level windows.

TYPE
Boolean

APPLICABILITY
I

62.30 Window.Top

NAME
Window.Top – set/get top edge of window

FUNCTION
Set the vertical position of the window. You can pass an absolute value in device-independent pixels here or use one of the following macros:

`Centered` Center window in the visible area of screen.

`Moused` Open window under the mouse cursor.

`Delta:<p>`
Open window `<p>` device-independent pixels below the screen's title bar.

Default for this tag is `Centered`.

TYPE
Number or predefined macro

APPLICABILITY
IG

62.31 Window.UseBottomBorderScroller

NAME
Window.UseBottomBorderScroller – enable bottom border scrollbar

PLATFORMS
AmigaOS and compatibles only

FUNCTION
Set this to `True` to tell RapaGUI that there is a child in your window layout which puts a scrollbar in the bottom window border, e.g. by using `Scrollbar.UseWinBorder`.

TYPE
Boolean

APPLICABILITY
I

62.32 Window.UseLeftBorderScroller

NAME
Window.UseLeftBorderScroller – enable left border scrollbar

PLATFORMS
AmigaOS and compatibles only

FUNCTION
Set this to `True` to tell RapaGUI that there is a child in your window layout which puts a scrollbar in the left window border, e.g. by using `Scrollbar.UseWinBorder`.

TYPE
Boolean

APPLICABILITY
I

62.33 Window.UseRightBorderScroller

NAME
Window.UseRightBorderScroller – enable right border scrollbar

PLATFORMS
AmigaOS and compatibles only

FUNCTION
Set this to `True` to tell RapaGUI that there is a child in your window layout which puts a scrollbar in the right window border, e.g. by using `Scrollbar.UseWinBorder`.

TYPE
Boolean

APPLICABILITY
I

62.34 Window.Title

NAME
Window.Title – set/get title of window

FUNCTION
Set or get the window's title.

If you do not set this attribute, RapaGUI will use the title specified in Hollywood's `@APPTITLE` preprocessor command. If `@APPTITLE` hasn't been specified either, the default title `"RapaGUI"` will be used.

TYPE
String

APPLICABILITY
ISG

62.35 Window.Width

NAME
Window.Width – set/get window width

FUNCTION
Set the window width in device-independent pixels. This can either be an absolute pixel value or one of the following special values:

`Default` Calculate width from the default sizes of all widgets.

`Screen:<1..100>`
Set width as a percentage of the host screen's total width.

Default for this tag is `Default`.

TYPE
Number or predefined macro

APPLICABILITY
IG

Appendix A Licenses

A.1 wxWidgets license

wxWindows Library Licence, Version 3.1

Copyright (c) 1998-2005 Julian Smart, Robert Roebling et al

Everyone is permitted to copy and distribute verbatim copies of this licence document, but changing it is not allowed.

WXWINDOWS LIBRARY LICENCE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public Licence as published by the Free Software Foundation; either version 2 of the Licence, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public Licence for more details.

You should have received a copy of the GNU Library General Public Licence along with this software, usually in a file named COPYING.LIB. If not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

EXCEPTION NOTICE

1. As a special exception, the copyright holders of this library give permission for additional uses of the text contained in this release of the library as licenced under the wxWindows Library Licence, applying either version 3.1 of the Licence, or (at your option) any later version of the Licence as published by the copyright holders of version 3.1 of the Licence document.

2. The exception is that you may use, copy, link, modify and distribute under your own terms, binary object code versions of works based on the Library.

3. If you copy code from files distributed under the terms of the GNU General Public Licence or the GNU Library General Public Licence into a copy of this library, as this licence permits, the exception does not apply to the code that you add in this way. To avoid misleading anyone as to the status of such modified files, you must delete this exception notice from such code and/or adjust the licensing conditions notice accordingly.

4. If you write modifications of your own for this library, it is your choice whether to permit this exception to apply to your modifications. If you do not wish that, you must delete the exception notice from such code and/or adjust the licensing conditions notice accordingly.

A.2 MUI license

This application uses MUI - MagicUserInterface (c) Copyright 1992-97 by Stefan Stuntz. MUI is a system to generate and maintain graphical user interfaces. With the aid of a preferences program, the user of an application has the ability to customize the outfit according to his personal taste.

MUI is distributed as shareware. To obtain a complete package containing lots of examples and more information about registration please look for a file called "muiXXusr.lha" (XX means the latest version number) on your local bulletin boards or on public domain disks.

If you want to pageview directly, feel free to send DM 30.- or US$ 20.- to

Stefan Stuntz
Eduard-Spranger-Straße 7
80935 München
GERMANY

Support and online registration is available at http://www.sasg.com/

A.3 Expat license

Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd and Clark Cooper

Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Expat maintainers.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.4 LGPL license

GNU Lesser General Public License Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original 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, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary 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, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Index

A

B

C

D

F

G

H

I

L

M

P

R

S

T

V

W