B
Parameters (Core JElement)

This appendix defines the classes used to handle parameters and details those parameters for Joomla! core elements. It contains the following classes:

- `JElement`
- `JParameter`
- Parameters

JElement

abstract, extends `JObject`*, located in* `/joomla/html/parameter/element.php`

This is an abstract class that is used to aid integration of extensions into Joomla!. A core use of this class enables the selection of custom parameter options when creating new menu items. The class is used in conjunction with an XML definition of an element, and used extensively by the `JParameter` class.

Direct descendents

`JElementCalendar`	Renders a calendar element
`JElementCategory`	Renders a category element
`JElementEditors`	Renders an editor element
`JElementFileList`	Renders a file list element
`JElementFolderList`	Renders a folder list element
`JElementHelpsites`	Renders a help site element
`JElementHidden`	Renders a hidden element
`JElementImageList`	Renders an image list element
`JElementLanguages`	Renders a languages element
`JElementList`	Renders a list element
`JElementMenu`	Renders a menu element
`JElementMenuItem`	Renders a menu item element
`JElementPassword`	Renders a password element
`JElementRadio`	Renders a radio button element

`JElementSection`	Renders a section element
`JElementSpacer`	Renders a spacer element
`JElementSQL`	Renders a SQL element
`JElementText`	Renders a text element
`JElementTextarea`	Renders a textarea element
`JElementTimezones`	Renders a time zone element
`JElementUserGroup`	Renders a user group element

Properties

string `$_name` = null	Element name; set in final renderer classes
object `$_parent` = null	Parent object that instantiated the element

Inherited properties

Inherited from `JObject`:

- `JObject::_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JElement` object and sets the parent object.

JElement **__construct**([**$parent** = null])

- *object* **&$parent**: Parent object that instantiated the element

fetchElement

This method returns the rendered element. This method must be overridden in element subclasses. For example, a call to `JElementTextarea::fetchElement()` might produce the following output:

```
<input type="text" name="controlName[name]"
       id="controlNamename"
       value="value" class="text_area" size="20" />
```

Redefined in descendants:

- `JElementCalendar::fetchElement()`
- `JElementCategory::fetchElement()`
- `JElementEditors::fetchElement()`
- `JElementFileList::fetchElement()`
- `JElementFolderList::fetchElement()`
- `JElementHelpsites::fetchElement()`
- `JElementHidden::fetchElement()`
- `JElementImageList::fetchElement()`
- `JElementLanguages::fetchElement()`
- `JElementList::fetchElement()`
- `JElementMenu::fetchElement()`
- `JElementMenuItem::fetchElement()`
- `JElementPassword::fetchElement()`
- `JElementRadio::fetchElement()`
- `JElementSection::fetchElement()`
- `JElementSpacer::fetchElement()`
- `JElementSQL::fetchElement()`
- `JElementText::fetchElement()`
- `JElementTextarea::fetchElement()`
- `JElementTimezones::fetchElement()`
- `JElementUserGroup::fetchElement()`

string **fetchElement($name, $value, &$xmlElement, $control_name)**

- *string* **$name**: Element `name` and `id` suffix
- *string* **$value**: Element value content
- object **$xmlElement**: `JSimpleXMLElement` element definition
- *string* **$control_name**: Element `name` and `id` prefix
- *string*: Returns rendered element

fetchTooltip

This method returns a tooltip encapsulated in HTML label tags. Redefined in descendants:

- `JElementHidden::fetchTooltip()`
- `JElementSpacer::fetchTooltip()`

string **fetchElement**(**$label**, **$description**, **&$xmlElement**,

[**$control_name** = ''], [**$name** = ''])

- *string* **$label**: Tooltip content and title
- *string* **$description**: Title suffix
- object **$xmlElement**: `JSimpleXMLElement` element definition
- *string* **$control_name**: Tooltip `id` prefix
- *string* **$ name**: Tooltip `id` suffix
- *string*: Returns HTML tooltip

getName

This method returns the name of the element.

string **getName**()

- *string*: Returns the element name

render

This method returns an array containing the rendered parts and attributes of the element. The array contains six items in order, tooltip `[0]`, rendered input element `[1]`, description `[2]`, label `[3]`, value `[4]`, and name `[5]`.

array **render**(**&$xmlElement**, **$value**, [**$control_name** = 'params'])

- object **$xmlElement**: `JSimpleXMLElement` element definition
- *string* **$value**: Element value
- *string* **$control_name**: Control name
- *array*: Returns an array of rendered parts and attributes of the element

JParameter

extends `JRegistry`, *located in* `/joomla/html/parameter.php`

This class handles INI string parameters. This class is used in conjunction with `JElement` subclasses and XML files that define the nature of parameters. INI strings are used in database tables for values that do not have a specific field. An instance of the class can be used to handle multiple INI strings (with different XML definitions), using groups to separate each one. When dealing with one INI string, omitting the group will always use the default group, `'_default'`. For further information on using the `JParameter` class refer to—Chapter 5, *Component Design,* Chapter 6, *Module Design,* and Chapter 7, *Plugin Design.*

Properties

string `$_raw` = null	The raw params string
string `$_xml` = null	The XML params element
array `$_elements` = array()	Loaded elements
array `$_elementPath` = array()	Directories where element types can be stored

Inherited properties

Inherited from `JRegistry`:

- `JRegistry::$_defaultNameSpace`
- `JRegistry::$_registry`

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JRegistry`:

- `JRegistry::__construct()`
- `JRegistry::getInstance()`
- `JRegistry::getNameSpaces()`
- `JRegistry::getValue()`
- `JRegistry::loadArray()`
- `JRegistry::loadFile()`
- `JRegistry::loadINI()`

- `JRegistry::loadObject()`
- `JRegistry::loadXML()`
- `JRegistry::makeNameSpace()`
- `JRegistry::merge()`
- `JRegistry::setValue()`
- `JRegistry::toArray()`
- `JRegistry::toObject()`
- `JRegistry::toString()`
- `JRegistry::__clone()`

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JParameter` object, loads `$data`, and if specified, the XML setup file. Redefinition of `JRegistry::__construct()`.

JParameter **__construct($data, [$path** = ''**]**)

- *string* **$data**: The raw parms text
- *string* **$path**: The path to the XML setup file

addElementPath

This method adds a path or an array of paths where `JParameter` should search for element types. You may either pass a string or an array of directories. `JParameter` will search for element types in the order they are added to the array.

void **addElementPath**(**$path**)

- *mixed* **$path**: Directory or directories to search
- *void*: No return

bind

This method binds data with parameters to the specified group. `$data` can be an associative array, an object, or INI string.

boolean **bind**(**$data**, [**$group** = '_default'**]**)

- *mixed* **$data**: The data to bind
- *string* **$group**: The parameter group to bind to the data
- *boolean*: Returns `true` if data was successfully bound

def

This method sets the default value for a parameter, in a specified group, if it is not already defined.

string **def**(**$key**, [**$default** = ''**]**, [**$group** = '_default'])

- *string* **$key**: The name of the parameter
- *string* **$default**: The default value of the parameter
- *string* **$group**: The parameter group to modify
- *string*: Returns the set value

get

This method gets the value of a parameter in a specified group. Returns the default value if the parameter is not set.

Redefines `JObject::get()` : returns a property of the object or the default value if the property is not set.

string **get($key**, **[$default** = ''**]**, **[$group** = '_default'**]**)

- *string* **$key**: The name of the parameter
- *string* **$default**: The default value of the parameter if not found
- *string* **$group**: The parameter group
- *string*: Returns the parameter value

getGroups

This method returns an associative array of the group names and the number of parameters in each as defined by the corresponding `JElement` object. Groups that do not have a `JElement` object will not be included.

mixed **getGroups**()

- *mixed*: Returns an associative array or `false` if no groups exist

getNumParams

This method returns the number of parameters defined by the associated `JElement` object in the specified group. If no parameters exist it returns `false`.

mixed **getNumParams**(**[$group** = '_default'**]**)

- *string* **$group**: The parameter group to count
- *string*: Returns integer count of parameters or `false` in none exist

getParam

This method returns an array of parameter details from a group. The array contains six items in order: tooltip [0], HTML rendered string [1], description [2], label [3], value [4], and name [5].

array **getParam**(**&$node**, **[$control_name** = 'params'**]**, **[$group** = '_default'**]**)

- *object* **$node**: A `JElement` param tag node
- *string* **$control_name**: The control name
- *string* **$group**: The parameter group
- *array*: Returns an array of parameters from a group

getParams

This method returns a two-dimensional array of all the parameters in a group. The array contains six items in order: tooltip [0], HTML rendered string [1], description [2], label [3], value [4], and name [5].

mixed **getParams**([**$name** = 'params'**]**, [**$group** = '_default'**]**)

- *string* **$name**: The control name or the text area if no setup file is found
- *string* **$group**: The parameter group
- *mixed*: Returns a two-dimensional array of all the parameters from a group

loadElement

This method loads an instance of a `JElement` subclass object based on `$type`. If an instance of the specified type does not exist it will be created. If `$new` is `true` a new instance will be created, even if there is an existing instance. `JElement` subclass objects are not restricted to groups.

JElement **&loadElement**(**$type**, [**$new** = false**]**)

- *string* **$type**: The control name or the text area if no setup file found
- *boolean* **$new**: The parameter group
- *object*: Returns a reference to an instance of a `JElement` subclass object

loadSetupFile

This method builds a `JSimpleXMLElement` object from an XML setup file. The XML file can include the group name; if it does not the group `'_default'` will be assumed.

boolean **&loadSetupFile**(**$path**)

- *string* **$path**: The path to an XML setup file
- *boolean*: Returns `true` upon success

render

This method renders all the parameters within a group and returns an HTML string.

string **render**([**$name** = 'params'**]**, [**$group** = '_default'**]**)

- *string* **$name**: The control name or the text area if no setup file found
- *string* **$group**: The parameter group
- *string*: Returns an HTML string

renderToArray

This method renders all the parameters within a group and returns an associative array. If `$group` does not exist it returns `false`.

mixed **renderToArray**([**$name** = 'params'**]**, [**$group** = '_default'**]**)

- *string* **$name**: The control name or the text area if no setup file found
- *string* **$group**: The parameter group
- *mixed*: Returns an associative array of parameters; `false` if no group exists

set

This method sets the value of a parameter in a specified group. Redefines `JObject::set()`: modifies a property of the object creating it if it does not already exist.

string **set**(**$key**, [**$value** = ''**]**, [**$group** = '_default'**]**)

- *string* **$key**: The name of the parameter
- *string* **$value**: The value of the parameter
- *string* **$group**: The parameter group
- *string*: Returns the set parameter value

setXML

This method sets an XML definition; the group is extracted from the group attribute of the object.

void **setXML**(**&$xml**)

- *object* **$xml**: The `JSimpleXMLElement` object to add
- *void*: No return

Parameters

We can use the XML tag `<param>` to define different parameters. Every `<param>` tag must include the following attributes:

Attribute	Description
`description`	Description of the parameter
`label`	Human-readable name of the input; always translated by `JText`
`name`	Name of the input
`type`	Type of parameter; this relates to `JElement` subclasses
`default`	The default value (this does not work for all elements)

This is an example of a `<param>` tag:

```
<param name="title" type="text" default="My Title"
       label="Title" description="Title of page" size="30" />
```

When we define the type we are informing Joomla! which `JElement` subclass to use to render the parameter. There are a number of core `JElement` subclasses available to us, each of which has its own set of attributes that modify the rendered output.

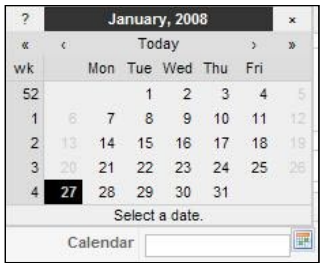
The following tables describe the parameter types from the core that we can use.

Any attributes that are optional are encapsulated in square braces. Remember that when we use the `param` tag we also need to include the attributes defined in the previous table.

calendar

The calendar parameter type provides a text box for entry of a date. An icon next to the text box provides a link to a pop-up calendar, which can also be used to enter the date value.

If the parameter has a saved value this is shown in the text box. Otherwise the default value, if any, is displayed.



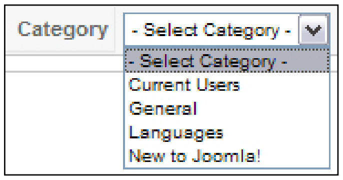
```
<param name="calendar" type="calendar" label="Calendar"
       default="05-20-2009" description="Select a date"
       class="inputbox" format="%m-%d-%Y" />
```

[class]	Optional; CSS Style, defaults to `'inputbox'` if omitted.
[default]	Optional; the default date, must be in the format described in **format**.
[format]	Optional; the format of the output date string (see *Chapter 12, Utilities and Useful Classes for syntax*.) Default is '%Y-%m-%d'.ed.

category

This element displays a drop-down selection box of published categories for a selected section.

The first option in the selection box is always `'- Select Category -'`; this option has a value of 0.



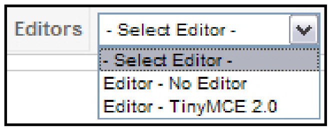
```
<param name="category" type="category" label="Category"
       default="1"
       description="Choose a category..."
       class="inputbox" section="3" />
```

[class]	Optional; CSS Style, defaults to `'inputbox'` if omitted.
[default]	Optional; the category ID number.
[section]	Optional; Section ID or component name if using component-specific categories. If not specified, all content categories are displayed.

editors

This element displays a drop-down selection box of available and enabled WYSIWYG editors.

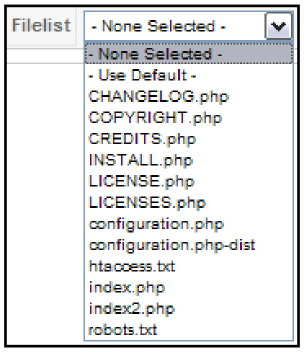
The first option in the selection box is always `'- Select Editor -'`; this option has a value of 0.



```
<param name="editor" type="editors"
       label="Default WYSIWYG Editor" default=""
       description="Editor for this User" />
```

filelist

This element displays a drop-down selection box of files in a specified directory:



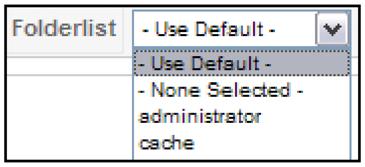
By default the first item on the list is `'- Do not use -'` which has a value of `'-1'`. The second item on the list is `'- Use default -'` which has a value of `'0'`.

```
<param name="layout" type="filelist" label="Filelist"
       description="Select a file" default=""
       directory="administrator"
       hide_default="" hide_none=""
       stripext="" filter="" exclude="" />
```

[Directory]	Optional; the filesystem path to the directory containing the files to be listed. If omitted the directory given by `JPATH_ROOT` is assumed.
[exclude]	Optional; a regular expression string which is used to exclude files from the list. The exclude argument expression is applied after the filter argument expression.
[filter]	Optional; a regular expression string which is used to filter the list of files selected for inclusion in the drop-down list. If omitted, all files in the directory are included. The filter argument expression is applied before the exclude argument expression.
[hide_default]	Optional; Boolean `true` hides the `'- Use default -'` option.
[hide_none]	Optional; Boolean `true` hides the `'- Do not use -'` option.
[stripExt]	Optional; Boolean `true` will remove extensions from the file list. If `true` then file name extensions will be stripped from the file names listed. Also note that the file name will be saved without the extension too.

folderlist

This element displays a drop-down selection box of folders in a specified directory:



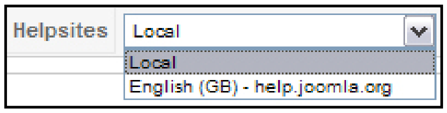
By default the first item on the list is `'- Do not use -'` which has a value of `'-1'`. The second item on the list is `'- Use default -'` which has a value of `'0'`.

```
<param name="folders" type="folderlist" label="Folderlist"
       description="Select a folder" default=""
       directory="administrator"
       hide_default="" hide_none=""
       filter="" exclude="" />
```

[directory]	The filesystem path to the directory containing the folders to be listed.
[exclude]	Optional; a regular expression string which is used to exclude folders from the list. The exclude argument expression is applied after the filter argument expression.
[filter]	Optional; a regular expression string which is used to filter the list of folders selected for inclusion in the drop-down list. If omitted, all folders in the directory are included. The filter argument expression is applied before the exclude argument expression.
[hide_default]	Optional; Boolean `true` hides the `'- Use default -'` option.
[hide_none]	Optional; Boolean `true` hides the `'- Do not use -'` option.

helpsites

This element displays a drop-down selection box of the help sites available for the specific installation:



With the exception of the "Local" entry, which is always added, the help sites are added from the `administrator/help/helpsites-1.5.xml` file.

```
<param name="helpsites" type="helpsites" label="Helpsites"
       default="" description="Select a helpsite" />
```

[default]	Optional; the default help site URL (not the visible help site name).

hidden

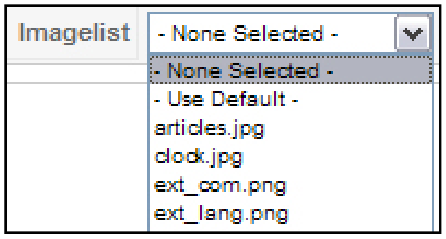
The hidden parameter type provides a field that has no visible control. A value cannot be assigned to it directly but can be assigned to it with code or by changing it in the `params.ini` file.

```
<param name="secretfield" type="hidden" default="secret"
       class="text_area" />
```

default	Mandatory; the value of the hidden field.
[class]	Optional; a CSS class name for the HTML form field. If omitted this will default to `'text_area'`.

imageList

This element displays a drop-down selection box of image files in a specified directory. Only files with `.png`, `.gif`, `.jpg`, `.bmp`, and `.ico` extensions are listed.



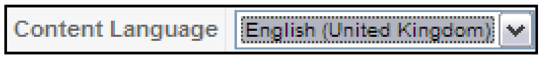
By default the first item on the list is `'- Do not use -'` which has a value of `'-1'`. The second item on the list is `'- Use default -'` which has a value of `'0'`.

```
<param name="images" type="imagelist" label="Imagelist"
       default="" description="Select an image"
       directory=""
       hide_default="" hide_none=""
       stripext="" exclude="" />
```

[directory]	Optional; the filesystem path to the directory containing the image files to be listed. If omitted the directory given by `JPATH_ROOT` is assumed.
[filter]	Optional; a regular expression string which is used to filter the list of image files selected for inclusion in the drop-down list. If omitted, all image files in the directory are included. The filter argument expression is applied before the exclude argument expression.
[exclude]	Optional; a regular expression string which is used to exclude image files from the list. The exclude argument expression is applied after the filter argument expression.
[hide_default]	Optional; Boolean `true` hides the `'- Use default -'` option.
[hide_none]	Optional; Boolean `true` hides the `'- Do not use -'` option.
[stripExt]	Optional; Boolean `true` will remove extensions from image file names listed. Also note that the file name will be saved without the extension as well.

languages

This element displays a drop-down selection box of known languages from a specific client:



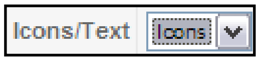
The first option on the list is always `'- Select Language -'` which has a value of `'0'`.

```
<param name="language" type="languages" default="en-GB"
       label="Content language" client="site"
       description="Choose a language"/>
```

[default]	Optional; the default language tag, for example- `en-GB`.
client	Mandatory; the application whose installed languages will be listed; use `site` for the frontend and `administrator` for the backend application.

list

This element displays a drop-down selection box of specified options:



The XML `<param>` element must include one or more `<option>` elements which define the list items. Don't forget to close the parameter definition with `</param>`. The text between the `<option>` and `</option>` tags is what will be shown in the drop down list and is a translatable string.

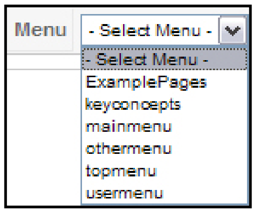
```
 <param name="contact_icons" type="list" default="0"
       label="Icons/text" description="Choose an icon"
   class="inputbox">
  <option value="0">Icons</option>
  <option value="1">Text</option>
  <option value="2">None</option>
</param>
```

[class]	Optional; CSS style, defaults to `'inputbox'` if omitted.
[default]	Optional; the default list item value.
[value]	Mandatory; value that will be saved for the parameter if the `<option>` is selected.

menu

This element parameter type provides a drop-down list of the available menus from your Joomla! site. If the parameter has a saved value this is selected when the page is first loaded. If not, the default value (if any) is selected.

The first option on the list is always `'- Select Menu -'` (which is a translatable string) and is given the value `'0'`.

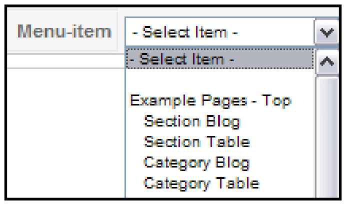


```
<param name="menu" type="menu" label="Menu"
       default="mainmenu" description="Select a menu" />
```

[default]	Optional; the default menu. Note that this is the name of the menu shown in the Type column on the Menu Manager screen and not the menu ID number.

menuitem

This element parameter type provides a drop-down list of the different menu items group by menu:



The first option on the list is always `'- Select Menu -'` (which is a translatable string) and is given the value `'0'`. The first item on the list will always have the `' - Top'` (which is a translatable string) appended to it.

```
<param name="menuitems" type="menuitem" label="Menu-item"
       default="45" description="Select a menu item"
       state="0" />
```

[default]	Optional; the default menu item. Note that this is the Item ID number of the menu item.
[state]	Optional; determines whether all menu items are listed or only published menu items. If `state = 0` then all menu items will be listed; if `state = 1` only published items will be listed.

password

This element parameter type provides a text box for entering a password; the text entered will be obscured by asterisks:

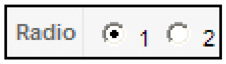
Note that the password string is stored in `params.ini` in cleartext; the stored value is not obscured by any hash function. Since most web servers will, by default, serve a `params.ini` file if the URL is entered in a web browser, this cannot be considered a secure method of holding a password.

```
<param name="password" type="password" label="Password"
       default="" description="Enter a password"
       class="inputbox" size="5" />
```

[class]	Optional; CSS style, defaults to `'text_area'` if omitted.
[default]	Optional; the default password.
[size]	Optional; Character width of the password box; if omitted the width is determined by the browser. The value of `size` does not limit the number of characters that can be entered.

radio

This element parameter type provides radio buttons to select different options:



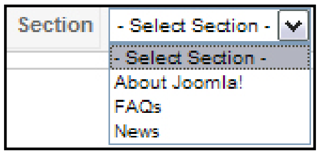
The XML `<param>` element must include one or more `<option>` elements which define the individual radio button items. Don't forget to close the parameter definition with `</param>`. The text between the `<option>` and `</option>` tags is what will be shown as the label for the radio button and is a translatable string. The first option is selected by default.

```
<param name="radio" type="radio" label="Radio"
       default="0" description="Enter a password">
  <option value="0">1</option>
  <option value="1">2</option>
</param>
```

[default]	Optional; the default radio item value.
value	Mandatory; the value that will be saved for the parameter if the `<option>` is selected.

section

This element displays a drop-down selection box of published sections:



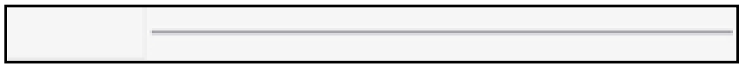
The first option in the selection box is always `'- Select Section -'`; this option has a value of 0.

```
<param name="section" type="section" label="Section"
      default="1" description="Choose a section" />
```

[default]	Optional; the default section ID number.

spacer

The spacer parameter type provides a visual separator (a horizontal rule) between parameter elements. It is a purely visible element and does not contain a value. The default horizontal rule can be replaced with text, including encoded html markup.



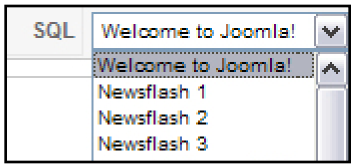
<param type="spacer" />

```
<param type="spacer" default="Title Text" />
<param type="spacer"
      default="&lt;b&gt;Title Text&lt;/b&gt;" />
```

[default]	Optional; a string which will be used instead of the `<hr />` that is inserted by default.

sql

This element parameter type displays a drop-down selection box of items determined by executing a query against the database.

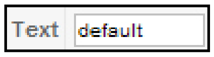


```
<param name="title" type="sql" label="SQL"
       default="" description="Select an item"
       query="SELECT id, title FROM #__content"
       key_field="id" value_field="title" />
```

name	The unique name of the parameter. This must match the name of the query results column that contains the values that will be displayed in the drop-down list.
[default]	Optional; this is the value of the `'value'` field, unless overridden by the `key_field` attribute.
[query]	The SQL query string which will provide the data for the drop-down list. The query must return two columns; one called `'value'` (unless overridden by the `key_field` attribute) which will hold the values of the list items; the other with the same as the value of the `name` attribute (unless overridden by the `value_field` attribute) containing the text to be shown in the drop-down list.
[key_field]	Optional; the name of the column that will contain values for the parameter; if omitted the `'value'` column will be used.
[value_field]	Optional; the name of the column that will contain values to be shown to the user in the drop-down list. If omitted then the column with the same name as the name attribute will be used, if it exists.

text

This element displays a text box for data entry:

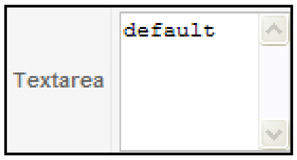


```
<param name="contact_icons" type="text" default="default"
       label="Text" description="Enter some text"
       class="text_area" size="10" />
```

[class]	Optional; CSS style, defaults to `'text_area'` if omitted.
[size]	Optional; Character width of the text box; if omitted the width is determined by the browser. The value of `size` does not limit the number of characters that can be entered.

textarea

This element displays a text box for multi-line data entry:

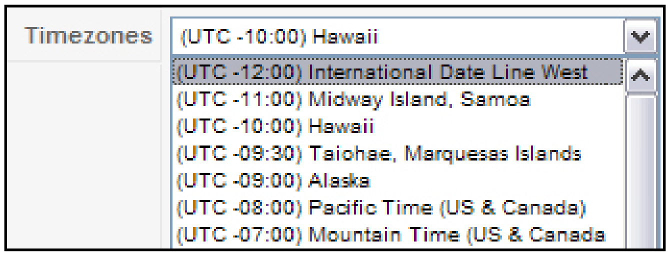


```
<param name="textarea" type="textarea" default="default"
       label="Textarea" description="Enter some text"
       class="text_area" rows="10" cols="5" />
```

[class]	Optional; CSS style, defaults to `'text_area'` if omitted.
[rows]	The height of the visible text area in lines. If omitted the height is determined by the browser. The value of rows does not limit the number of lines that may be entered.
[cols]	The width of the visible text area in characters. If omitted the width is determined by the browser. The value of cols does not limit the number of characters that may be entered.

timezones

This element displays a drop-down selection box of different time zones. Values are identified as plus or minus hours from **UTC** (Universal Time Code); **UTC** is the same as **GMT** (Greenwich Mean Time) and **Z** (Zulu Time).

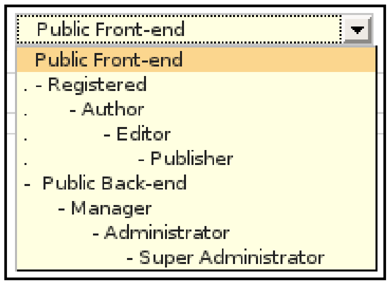

Timezones
(UTC -10:00) Hawaii
(UTC -12:00) International Date Line West
(UTC -11:00) Midway Island, Samoa
(UTC -10:00) Hawaii
(UTC -09:30) Taiohae, Marquesas Islands
(UTC -09:00) Alaska
(UTC -08:00) Pacific Time (US & Canada)
(UTC -07:00) Mountain Time (US & Canada

```
<param name="timezone" type="timezones" default="-10"
       label="Timezones" description="Select a timezone" />
```

[default]	The default timezone offset (for example -10 is Hawaii).

usergroup

This parameter element type provides a drop-down list of user groups:



```
<param name="usergroup" type="usergroup" default="1"
       label="User Groups" description="Select a user group"
       class="inputbox" size="10" multiple="yes" />
```

[class]	Optional; CSS style, defaults to `'inputbox'` if omitted.
[default]	Optional; the default user group id number.
[size]	Optional; the number of rows of the list to display. If there are more items in the list than the size specified then scroll-bars will appear. If there is no size attribute, the list will be shown as a drop-down list if multiple is not set. If multiple is true and no size is set, the default size is the number of items in the list.
[multiple]	Optional; indicates whether multiple selections are allowed. If the multiple attribute is present (for example, multiple="yes"), then multiple items may be selected from the drop-down list. If omitted, then only one item may be selected. Note that if multiple is used, the selected values are returned as an array. Also, if multiple is used, you should set the size attribute to control the size of the list box.

C
Registry and Configuration

This appendix details the Joomla! registry and site settings we normally would expect to be present in the configuration namespace. The appendix covers:

- JRegistry
- Site Configuration Settings

JRegistry

extends `JObject`, *located in* `/joomla/registry/registry.php`

This class handles configuration details in a hierarchy using namespaces. For further information on using the `JRegistry` class refer to Chapter 4, *Extension Design*.

Direct descendents

`JParameter`	Class to handle parameters

Properties

string `$_defaultNameSpace` = null	The default namespace
array `$_registry` = array()	An array of namespace objects

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JRegistry` object. Adds the namespace and sets it as the default. Redefinition of `JObject::__construct()`; class constructor, overridden in descendant classes. Redefined in descendant `JParameter::__construct()` constructor.

JRegistry **__construct**([**$namespace**= 'default'])

- *string* **$namespace**: The default registry namespace; optional

getInstance

This method returns a reference to a global JRegistry object. If the registry object does not exist, it creates it. `$namespace` is optional and is only used if the instance does not exist. This method must be invoked as follows:

```
$registry =& JRegistry::getInstance($id, $namespace);
```

JRegistry **&getInstance**(**$id**, [**$namespace** = 'default'])

- *string* **$id**: An ID for the registry instance
- *string* **$namespace**: Optional registry default namespace
- *object*: Returns a reference to a global `JRegistry` object

getNameSpaces

This method returns an array containing all of the namespaces in the registry.

array **getNameSpaces**()

- *array*: Returns an array of all the registry namespaces

getValue

This method returns a value from the registry. `$regpath` can include multiple levels separated by periods. If the path includes no periods, the value will be retrieved from the default namespace. If the value is not set, the default value will be returned.

mixed **getValue**(**$regpath**, [**$default** = null])

- *string* **$regpath**: Registry path (for example `joomla.content.showauthor`)
- *mixed* **$default**: Optional default value
- *mixed*: Returns a registry entry value or the default value

loadArray

This method loads an associative array into the registry namespace. The `$array` keys must not contain periods. If the namespace is not specified the default namespace will be used.

boolean **loadArray**(**$array**, [**$namespace** = null])

- *array* **$array**: An associative array of values to load
- *string* **$namespace**: Optional namespace; default will be used if missing
- *boolean*: Returns `true` upon success

loadFile

This method loads a configuration file into the registry namespace. The `$file` parameter keys must not contain periods. Possible formats are `INI`, `PHP`, and `XML`. If the namespace is not specified the default namespace will be used.

boolean **loadFile(**$**file**, [**$format** = 'INI'**]**, [**$namespace** = null**]**)

- *string* **$file**: The path (including file name) to the file to load
- *string* **$format**: Optional format of the file; defaults to `'INI'`
- *string* **$namespace**: Optional namespace; default will be used if missing
- *boolean*: Returns `true` upon success

loadINI

This method loads an INI string into the registry namespace. The `$data` parameter keys must not contain periods. If the namespace is not specified the default namespace will be used.

boolean **loadINI(**$**data**, [**$namespace** = null**]**)

- *string* **$data**: The INI formatted string to load into the registry
- *string* **$namespace**: Optional namespace; default will be used if missing
- *boolean*: Returns `true` upon success

loadObject

This method loads the public properties of the object into the registry namespace. If the namespace is not specified the default namespace will be used.

boolean **loadObject(&$object**, [**$namespace** = null**]**)

- *object* **$object**: The object whose properties to load into the registry
- *string* **$namespace**: Optional namespace; default will be used if missing
- *boolean*: Returns `true` upon success

loadXML

This method loads an XML string into the registry namespace. The `$data` parameter keys must not contain periods. If the namespace is not specified the default namespace will be used.

boolean **loadXML($data**, **[$namespace** = null**]**)

- *string* **$data**: The XML formatted string to load into the registry
- *string* **$namespace**: Optional namespace; default will be used if missing
- *boolean*: Returns `true` upon success

makeNameSpace

This method creates a new namespace in the registry. If the namespace already exists it will be overwritten.

boolean **makeNameSpace**(**$namespace**)

- *string* **$namespace**: The name of the namespace to create
- *boolean*: Returns `true` upon success

merge

This method merges the registry data with `$source` registry data. The `$source` values take precedence over existing values. If a namespace does not exist it will be created.

boolean **merge**(**&$source**)

- *JRegistry* **$source**: The source `JRegistry` object to merge
- *boolean*: Returns `true` upon success

setValue

This method sets a value in the registry. The `$regpath` can include multiple levels separated by periods. If the path includes no periods, the value will be set in the default namespace. If the `$regpath` does not exist it will be created.

mixed **setValue**(**$regpath**, **$value**)

- *string* **$regpath**: The registry path (for example `joomla.content.showauthor`)
- *mixed* **$value**: The registry entry value
- *mixed*: Returns the previous value or `false` upon failure

toArray

This method transforms a registry namespace into an associative array. If `$namespace` is not specified the default namespace will be used.

array **toArray**([**$namespace** = null**]**)

- *string* **$namespace**: Optional namespace; default will be used if missing
- *array*: Returns an associative array representation of the registry namespace

toObject

This method transforms a registry namespace into a `stdClass` object. If `$namespace` is not specified the default namespace will be used.

array **toObject**([**$namespace** = null**]**)

- *string* **$namespace**: Optional namespace; default will be used if missing
- *array*: Returns `stdClass` object representation of the registry namespace

toString

This method transforms a registry namespace into a string in the specified format. Possible formats are `INI`, `PHP`, and `XML`. If the namespace is not specified the default namespace will be used. `$params` is passed to the format handler `objectToString()` method; the contents of `$params` depends upon the format. Some format handlers are restricted to a maximum depth. Redefines `JObject::toString()` : object-to-string conversion.

string **toString**([**$format** = 'INI'**]**, [**$namespace** = null**]**, [**$params** = null**]**)

- *string* **$format**: The return format of the string
- *string* **$namespace**: Optional namespace; default will be used if missing
- *mixed* **$params**: Formatter parameters (see formatters for more info)
- *string*: Returns the registry namespace in the specified format

_clone

This method creates a clone of the registry.

void **_clone**()

- *void*: No return types

Site configuration settings

The config namespace is located in the registry; most of the settings originate from the `configuration.php` file. This table lists the common settings found within the `configuration.php` file:

Name	Description
absolute_path	Full path to the Joomla! installation, for example `/www/joomla`
cache_handler	Mechanism with which to handle caching; Joomla! supports `APC`, `EAccelerator`, `memcache`, and `file`
cachetime	Cache life expectancy in seconds
caching	Caching enabled; `1=enabled`, `0=disabled`
db	Database name
dbprefix	Database table prefix
dbtype	Database driver
debug	Site debug status; `1=enabled`, `0=disabled`
debug_db	Database debug status; `1=enabled`, `0=disabled`
debug_lang	Language debug status; `1=enabled`, `0=disabled`
editor	Default editor
error_reporting	Error reporting level: `-1=system default`, `0=none`, `7=simple`, `2047=maximum`
feed_email	Email address of the feed author
feed_limit	Number of content feed items to display
feed_summary	Display full text in feeds; `1=true`, `0=false`
force_ssl	Force areas of the site to be SSL ONLY. `0=None`, `1=Administrator`, `2=Both Site and Administrator`
fromname	Mail email address alias (see *mailfrom*)
ftp_enable	FTP access enabled: `1=true`, `0=false`
ftp_host	FTP host, normally `127.0.0.1`
ftp_pass	FTP account password
ftp_port	FTP port, normally `21`
ftp_root	FTP path to Joomla! installation
ftp_user	FTP account username
gzip	GZIP compression enabled: `1=true`, `0=false`
helpurl	Joomla! help site
host	Host name
lang	Default language name
lang_administrator	Default backend language tag
lang_site	Default frontend language tag

Name	Description
language	Default language tag
lifetime	Session lifetime in minutes
list_limit	Default length of lists (pagination) in the backend
live_site	URI to the site
log_path	Path to the site LOG files
mailer	Email sending mechanism; Joomla! supports: PHP mail, sendmail, and SMTP
mailfrom	Default email sender address
memcache_settings	Settings for Memcache (serialized PHP data); Memcache is a PHP caching system
metaAuthor	Global option to show the author's meta tag when viewing a content item: `1=true`, `0=false`
metaDesc	Site metadata-description-tag content: `'Joomla! - the dynamic portal engine and content management system'`
metaKeys	Site metadata keys tag content: `'joomla, Joomla'`
metaTitle	Display the site metadata title tag: `1=show`, `0=hide`
offline	Site offline: `1=true`, `0=false`
offline_message	`This site is down for maintenance. Please check back again soon.`
offset	Time zone hours offset from UTC (also known as GMT and Z)
password	Database account password
secret	Secret site word (random alphanumeric string)
sef	SEF enabled: `1=true`, `0=false`
sef_rewrite	Apache SEF mod_rewrite enabled: `1=true`, `0=false`
sef_suffix	Add a suffix to the url based on the document type if `sef_suffix=1`
sendmail	`/usr/sbin/sendmail`
session_handler	Session storage handling mechanism; Joomla! supports: `APC`, `database`, `EAccelerator`, and `Memcache`
sitename	Name of the Joomla! installation
smtpauth	SMTP requires authorization: `1=true`, `0=false`
smtphost	SMTP host, normally `localhost`
smtppass	SMTP account password
smtpuser	SMTP account username
tmp_path	Temporary directory; used for archive extraction
user	Database account username
xmlrpc_server	XMLRPC support: `1=enabled`, `0=disabled`

D
Menus and Toolbars

There are many menu bar buttons available through the `JToolbarHelper` class; custom buttons can also be defined. These buttons come with pre-defined behaviors which simplify the effort required to implement their functionality. This appendix covers:

- `JMenu`
- `JPathway`
- Toolbar buttons

JMenu

extends `JObject`, *located in* /`joomla/application/menu.php`

This class handles menus and menu items. For further information on using the `JMenu` class refer to Chapter 9, *Customizing the Page*.

Properties

integer `$_active` = 0	Identifier of the active menu item
integer `$_default` = 0	Identifier of the default menu item
array `$_items`	Array of menu items (`stdClass` objects)

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JMenu` object. Redefinition of `JObject::__construct()`; overridden in descendant classes.

JMenu **__construct**([**$options**])

- *array* **$options**: Menu options

authorize

This method checks the `JMenu` object authorization against an access control object and optionally an access extension object.

boolean **authorize**(**$id**, **$accessid**)

- *integer* **$id**: The menu id
- *integer* **$accessid**: The user access identifier
- *boolean*: Returns `true` if authorized to view the menu item

getActive

This method returns a reference to the current/active menu item or if the current item is not set returns the default menu item.

object **&getActive**()

- *object*: Returns a reference to the current/active menu item object

getDefault

This method returns a reference to the default menu item (home page).

object **&getDefault**()

- *object*: Returns a reference to the default menu item object

getInstance

This method returns a reference to the global `JMenu` object. If the menu object does not exist it creates it. The method must be invoked as:

```
$menu = &JMenu::getMenu();
```

JMenu **&getInstance**(**$client**, [**$options** = array()])

- *string* **$client**: The name of the the client.
- *array* **$options**: An associative array of options.
- *object*: Returns a reference to a `JMenu` object.

getItem

This method returns a reference to a menu item based on `$id`. If the menu item does not exist it returns null.

mixed **&getItem**(**$id**)

- *integer* **$id**: The menu item id
- *mixed*: Returns a reference to menu item object or null

getItems

This method returns an active menu item or an array of menu items based on an attribute. The returned active menu item attributes must match the specified `$attribute` and `$value`. If `$firstonly` is `true` then the method retrieves the first matching menu item.

mixed **getItems**(**$attribute**, **$value**, [**$firstonly** = false])

- *string* **$attribute**: The attribute to check
- *string* **$value**: The value to compare the attribute against
- *boolean* **$firstonly**: If true, only return the first matching menu item
- *mixed*: Returns a menu item or an array of menu items

getMenu

This method returns an array of all the menu items.

array **getMenu**()

- *array*: Returns an array of all the menu items

getParams

This method returns a reference to a `JParameter` object that contains all the parameters for a specified menu item. If the menu item does not exist or is not published it returns an empty `JParameter` object.

JParameter **&getParams**(**$id**)

- *integer* **$id**: The menu item id
- *object*: Returns a reference to a `JParameter` object

load

This *abstract* method loads the menu items.

array **load**()

- *array*: Returns an array of all the menu items

setActive

This method sets the active menu item by `$id`.

mixed **&setActive($id)**

- *integer* **$id**: The menu item id
- *mixed*: Returns a reference to a `JMenu` object or null

setDefault

This method sets the default menu item by `$id`.

boolean **&setDefault($id)**

- *integer* **$id**: The menu item id
- *boolean*: Returns `true` upon success; `false` upon failure

JPathway

abstract, extends `JObject`*, located in* /`joomla/application/pathway.php`

This is an abstract class that handles breadcrumbs. This class is used to model the breadcrumb trail that is used in most templates as a way of describing a user's current position within a site. For further information on using the `JPathway` class refer to Chapter 9, *Customizing the Page*.

Properties

integer `$_count` = 0	The number of breadcrumbs
array `$_pathway` = null	An array of pathway item objects

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JPathway` object. Redefinition of `JObject::__construct()`; class constructor, overridden in descendant classes.

JPathway **__construct**([**$options**= array()**]**)

- *array* **$options**: An associative array of values

addItem

This method adds a breadcrumb to the end of the pathway.

boolean **addItem(\$name**, **[\$link** = ''**]**)

- *string* **$name**: Name of the breadcrumb
- *string* **$link**: Breadcrumb URI
- *boolean*: Returns `true` upon success

getInstance

This method returns a reference to a `JPathway` object. The method must be invoked as:

```
$menu = $Jpathway::getInstance();
```

JPathway **&getInstance($client**, **[$options** = array()**])**

- *string* **$client**: The name of the client
- *string* **$options**: An associative array of options
- *object*: Returns a reference to a `JPathway` object

getPathway

This method returns an array of breadcrumbs. Breadcrumbs are represented as `stdClass` objects with two properties: `name` and `link`.

array **getPathway**()

- *array*: Returns an array of breadcrumbs in the order of their display

getPathwayNames

This method returns an array of breadcrumb names.

array **getPathwayNames**()

- *array*: Returns an array of breadcrumb names in the order of their display

setItemName

This method sets the name of a breadcrumb. `$id` refers to the breadcrumb number; breadcrumb ids are zero-based.

boolean **setItemName($id**, **$name**)

- *integer* **$id**: Breadcrumb number
- *string* **$name**: Breadcrumb name
- *boolean*: Returns `true` upon success

setPathway

This method sets the `JPathway` items array.

array **setPathway**(**$pathway**)

- *string* **$pathway**: An array of `JPathway` objects
- *array*: Returns the previous array of `JPathway` items

_makeItem

This method creates and returns a new breadcrumb `stdClass` object.

object **_makeItem**(**$name**, **$link**)

- *string* **$name**: Name of the new breadcrumb
- *string* **$link**: Breadcrumb URI
- *object*: Returns a new breadcrumb `stdClass` object

Toolbar buttons

Toolbar buttons are created using `JToolBarHelper` methods. For further information on using the `JToolBarHelper` class refer to Chapter 8, *Rendering Output*. The following tables provide further detail on the available buttons:

This places an 'add new' button on the menu bar. The name displays under the icon.

JToolBarHelper::**addNew**($**task** = 'add', $**alt** = 'New');

- *string* **task**: Optional; the task to perform. Default is `add`.
- *string* **alt**: Optional; the display name. Default is `New`.

This places an 'add new' button on the menu bar. The name displays under the icon. This method hides/disables the main menu when pressed; the `adminForm` form must include a hidden input field called `hidemainmenu`.

JToolBarHelper::**addNewX**($**task** = 'add', $**alt** = 'New');

- *string* **task**: Optional; the task to perform. Default is `add`.
- *string* **alt**: Optional; the display name. Default is `New`.

This places an apply button on the menu bar. The name displays under the icon.

JToolBarHelper::**apply**($**task** = 'apply', $**alt** = 'Apply');

- *string* **task**: Optional; the task to perform. Default is `apply`.
- *string* **alt**: Optional; the display name. Default is `Apply`.

This places an archive button on the menu bar. The name displays under the icon.

JToolBarHelper::**archiveList**($**task** = 'archive', $**alt** = 'Archive');

- *string* **task**: Optional; the task to perform. Default is `archive`.
- *string* **alt**: Optional; the display name. Default is `Archive`.

This places an assign button on the menu bar. The name displays under the icon.

JToolBarHelper::**assign**($**task** = 'assign', **alt**='Assign');

- *string* **task**: Optional; the task to perform. Default is `assign`.
- *string* **alt**: Optional; the display name. Default is `Assign`.

This places a back button on the menu bar. The name displays under the icon.

JToolBarHelper::**back**($**task** = 'apply', $**href** = 'javascript:history.back();');

- *string* **alt**: Optional; the display name. The default is `Back`.
- *string* **href**: Optional; URI string.

This places a cancel button on the menu bar. The name displays under the icon.

JToolBarHelper::**cancel**($**task** = 'cancel', $**alt** = 'Cancel');

- *string* **task**: Optional; the task to perform. Default is `cancel`.
- *string* **alt**: Optional; the display name. Default is `Cancel`.

This places a custom button on the menu bar. The name displays under the icon. If `x` is `true` the method hides/disables the main menu when pressed; the `adminForm` form must include a hidden input field called `hidemainmenu`.

JToolBarHelper::**custom**($**task** = '', $**icon** = '', $**iconOver** = '',
$**alt** = '', $**listSelect** = true, $**x** = false);

- *string* **task**: The task to perform
- *string* **icon**: The icon to use
- *string* **iconOver**: The icon to use on mouse over
- *string* **alt**: Optional; the display name
- *boolean* **listSelect**: If `true` check if a list item is selected
- *boolean* **x**: If `true` hide/disable main menu

This places a custom button on the menu bar. The name displays under the icon. This method hides/disables the main menu when pressed; the `adminForm` form must include a hidden input field called `hidemainmenu`.

JToolBarHelper::**customX**($**task** = '', $**icon** = '', $**iconOver** = '',
$**alt** = '', $**listSelect** = true);

- *string* **task**: The task to perform
- *string* **icon**: The icon to use
- *string* **iconOver**: The icon to use on mouse over
- *string* **alt**: Optional; the display name
- *boolean* **listSelect**: If `true` check if a list item is selected

This places a delete button on the menu bar. The name displays under the icon.

JToolBarHelper::**deleteList**($**msg** = '', $**task** = 'remove', $**alt** = 'Delete');

- *string* **msg**: Optional; delete confirmation message.
- *string* **task**: Optional; the task to perform. Default is `remove`.
- *string* **alt**: Optional; the display name. Default is `Delete`.

This places an assign button on the menu bar. The name displays under the icon.

JToolBarHelper::**deleteListX**($**msg** = '', $**task** = 'remove', $**alt** = 'Delete');

- *string* **msg**: Optional; delete confirmation message.
- *string* **task**: Optional; the task to perform. Default is `remove`.
- *string* **alt**: Optional; the display name. Default is `Delete`.

This places a divider, a vertical line, on the menu bar to visually separate groups of buttons.

JToolBarHelper::**divider**();

This places an 'edit css' button on the menu bar. The name displays under the icon.

JToolBarHelper::**editCss**($**task** = 'edit_css', $**alt** = 'Edit CSS');

- *string* **task**: Optional; the task to perform. Default is `edit_css`.
- *string* **alt**: Optional; the display name. Default is `Edit CSS`.

This places an 'edit css' button on the menu bar. The name displays under the icon. This method hides/disables the main menu when pressed; the `adminForm` form must include a hidden input field called `hidemainmenu`.

JToolBarHelper::**editCssX**($**task** = 'edit_css', $**alt** = 'Edit CSS');

- *string* **task**: Optional; the task to perform. Default is `add`.
- *string* **alt**: Optional; the display name. Default is `New`.

This places an 'edit html' button on the menu bar. The name displays under the icon.

JToolBarHelper::**editHtml**($**task** = 'edit_source', $**alt** = 'Edit HTML');

- *string* **task**: Optional; the task to perform. Default is `edit_source`.
- *string* **alt**: Optional; the display name. Default is `Edit HTML`.

This places an 'edit html' button on the menu bar. The name displays under the icon. This method hides/disables the main menu when pressed; the `adminForm` form must include a hidden input field called `hidemainmenu`.

JToolBarHelper::**editHtmlX**($**task** = 'edit_source', $**alt** = 'Edit HTML');

- *string* **task**: Optional; the task to perform. Default is `edit_source`.
- *string* **alt**: Optional; the display name. Default is `Edit HTML`.

This places an 'edit' button on the menu bar. This button requires that at least one list item be selected, `cid[]`.The name displays under the icon.

JToolBarHelper::**editList**($**task** = 'edit', $**alt** = 'Edit');

- *string* **task**: Optional; the task to perform. Default is `edit`.
- *string* **alt**: Optional; the display name. Default is `Edit`.

This places an 'edit' button on the menu bar. This button requires that at least one list item be selected, `cid[]`.The name displays under the icon. This method hides/disables the main menu when pressed; the `adminForm` form must include a hidden input field called `hidemainmenu`.

JToolBarHelper::**editListX**($**task** = 'edit', $**alt** = 'Edit');

- *string* **task**: Optional; the task to perform. Default is `edit`.
- *string* **alt**: Optional; the display name. Default is `Edit`.

This places a 'help' button on the menu bar. `$ref` determines the help file to use. `$com` chooses to use a component specific help file located in the administrator component help folder. The name displays under the icon.

JToolBarHelper::**help**($**ref**, $**com** = false);

- *string* **ref**: The help file to use.
- *boolean* **com**: Optional; use component-specific help files. Default is `false`.

This places an 'make-default' button on the menu bar. The name displays under the button.

JToolBarHelper::**makeDefault**($**task** = 'default', $**alt** = 'Default');

- *string* **task**: Optional; the task to perform. Default is `default`.
- *string* **alt**: Optional; the display name. Default is `Default`.

This places a button on the menu bar that when pressed allows an administrator to upload a file to the media manager. The name displays under the icon.

JToolBarHelper::**media_manager** ($**directory** = '', $**alt** = 'Upload');

- *string* **directory**: Optional; where to place the file.
- *string* **alt**: Optional; the display name. Default is `Upload`.

This places a 'preferences' button on the menu bar. The name displays under the icon. When pressed a pop-up box appears with the component's preferences as defined by the XML file. If `path` is not specified the default location, `JPATH_COMPONENT_ADMINISTRATOR.'config.xml'`, is used.

JToolBarHelper::**preferences**($**component**, $**height** = '150', $**width** = '570', $**alt** = 'Preferences', $**path** = '');

- *string* **component**: The component name.
- *integer* **height**: Optional; the popup box height. Default is `'150'`.
- *integer* **width**: Optional; the popup box width. Default is `'570'`.
- *string* **alt**: Optional; the display name.
- *string* **path**: Optional; the path to the configuration XML file.

This places a 'preview' button on the menu bar and appends `&task=preview` to the URI. The name displays under the icon.

JToolBarHelper::**preview** ($**url** = '', $**updateEditors** = false);

- *string* **task**: Optional; the URI
- *boolean* **updateEditors**: Deprecated

This places a 'publish' button on the menu bar. The name displays under the icon.

JToolBarHelper::**publish**($**task** = 'publish', $**alt** = 'Publish');

- *string* **task**: Optional; the task to perform. Default is `publish`.
- *string* **alt**: Optional; the display name. Default is `Publish`.

This places a 'publish' button on the menu bar. This button requires that at least one list item be selected, `cid[]`.The name displays under the icon.

JToolBarHelper::**publishList**($**task** = 'edit', $**alt** = 'Edit');

- *string* **task**: Optional; the task to perform. Default is `publish`.
- *string* **alt**: Optional; the display name. Default is `Publish`.

This places a 'save' button on the menu bar. The name displays under the icon.

JToolBarHelper::**save**($**task** = 'save', $**alt** = 'Save');

- *string* **task**: Optional; the task to perform. Default is `save`.
- *string* **alt**: Optional; the display name. Default is `Save`.

This places a spacer on the menu bar; use the width parameter to determine the size of the spacer.

JToolBarHelper::**spacer**($**width** = '');

- *integer* **width**: The spacer width

This sets the title and the icon title class of the menu bar.

JToolBarHelper::**title**($**title**, $**icon** = 'generic.png');

- *string* **title**: The title on the menu bar.
- *string* **icon**: Optional; the title class, prepended to `icon-48-`

This places a 'trash' button on the menu bar. The name displays under the icon.

JToolBarHelper::**trash**($**task** = 'remove', $**alt** = 'Trash', $**check** = true);

- *string* **task**: Optional; the task to perform. Default is `remove`
- *string* **alt**: Optional; the task to perform. Default is `Trash`
- *boolean* **check**: Optional; check that an item is selected. Default is `true`

This places a 'unarchive' button on the menu bar. This button requires that at least one list item be selected, `cid[]`. The name displays under the icon.

JToolBarHelper::**unarchiveList**($**task** = 'unarchive', $**alt** = 'Unarchive');

- *string* **task**: Optional; the task to perform. Default is `save`.
- *string* **alt**: Optional; the display name. Default is `Save`.

This places an 'unpublish' button on the menu bar. The name displays under the icon.

JToolBarHelper::un**publish**($**task** = 'unpublish', $**alt** = 'Unpublish');

- *string* **task**: Optional; the task to perform. Default is `unpublish`.
- *string* **alt**: Optional; the display name. Default is `Unpublish`.

This places an 'unpublish' button on the menu bar. This button requires that at least one list item be selected, `cid[]`. The name displays under the icon.

JToolBarHelper::**unpublishList**($**task** = 'unpublish', $**alt** = 'Unpublish');

- *string* **task**: Optional; the task to perform. Default is `unpublish`.
- *string* **alt**: Optional; the display name. Default is `Unpublish`.

E
Joomla! HTML Library

Joomla! provides a comprehensive library of classes to render XHTML. The library contains classes to render HTML elements, parameters, toolbars, and editors. It also includes classes to handle list filtering, ordering, and pagination. In other words, the `joomla.html` library provides the tools to easily render output.
This appendix details:

- The joomla.html library
- Basic element types (JHTML)
    - Calendar
    - Date
    - iframe
    - Image
    - Link
    - Script
    - Style
    - Tooltip
- Complex element types
    - JHTMLBehavior
    - JHTMLEMail
    - JHTMLForm
    - JHTMLGrid
    - JHTMLImage
    - JHTMLList
    - JHTMLMenu
    - JHTMLSelect

- JPane
- JPaneSliders
- JPaneTabs

The joomla.html library

This part of the library is used to aid in the rendering of `XHTML` and consists of the static `JHTML` class, which supports eight basic HTML element types, and eight classes that support more complex elements.

Basic element types

All of the element types, whether basic or complex, are managed using the class loader method `JHTML::_()`. The basic types are:

`calendar`	Generates a calendar control field and a clickable calendar image
`date`	Returns formatted date string
`iframe`	Generates a XHTML `<iframe></iframe>` element
`image`	Generates a XHTML `<img></img>` element
`link`	Generates a XHTML `<a></a>` element
`script`	Generates a XHTML `<script></script>` element
`style`	Generates a `<link rel="stylesheet" style="text/css" />` element
`tooltip`	Generates a popup tooltip using JavaScript

JHTML

static, located in `/joomla/html/html.php`

This is a Joomla! utility class used to aid in the rendering of XHTML. For more information about the `JHTML` class refer to Chapter 8, *Rendering Output*.

Methods

_(Class Loader Method)

This method loads HTML sub-classes. The `$type` can consist of up to three parts connected with dots (.): `prefix.class.function`. The default prefix is `JHTML` and is only needed to load custom HTML helper classes. The class part identifies the sub-class file (for example, behavior) and the function identifies the method. Additional parameters may be provided and passed on to the sub-class.
For example:

```
echo JHTML::_('tooltip', 'content', 'title');
```

There are eight basic types that are executed from the `JHTML` class: `calendar`, `date`, `iframe`, `image`, `link`, `script`, `stylesheet`, and `tooltip`. There are eight grouped types: `behavior`, `email`, `form`, `grid`, `image`, `list`, `menu`, and `select`. Grouped types are separate classes with their own methods. To execute a method within one of the group types you use the `class.function` for `$type` for example. `email.cloak` or `form.token`). Further information on the basic and grouped types can be found in Chapter 8, *Rendering Output*.

mixed _(**$type**)

- *string* **$type**: sub-class method to execute
- *mixed*: Returns the result of sub-class method execution or `false` on failure

addIncludePath

This method adds a directory where the JHTML class should search for helpers. `$path` may be a string or an array of directory names.

array **addIncludePath**(**$path** = '')

- *mixed* **$path**: A path or array of paths to add
- *array*: Returns an array of path names

calendar

This method generates the HTML to display a calendar control.

string **calendar**(**$value**, **$name**, **$id**, [**$format** = '%Y-%m-%d'], [**$attribs** = null])

- *string* **$value**: The date value
- *string* **$name**: The name of the text field
- *string* **$id**: The id of the text field
- *string* **$format**: The date format
- *array* **$attribs**: Additional HTML attributes
- *string*: Returns XHTML string

date

This method takes a `$date` and formats it accordingly. The `$date` should always be UTC. The `$offset` is retrieved from the registry unless a custom `$offset` is provided.

string **date**(**$date**, [**$format** = null], [**$offset** = null])

- *string* **$date**: The date value
- *string* **$format**: Optional format string
- *integer* **$offset**: Optional offset; default is configuration offset
- *string*: Returns formatted date string

iframe

This method generates the HTML to display a `<iframe></iframe>` element.

string **iframe**(**$url**, **$name**, [**$attribs** = null], [**$noFrames** = ''])

- *string* **$url**: The relative URL to use for the `src` attribute
- *string* **$name**: The target attribute to use
- *array* **$attribs**: An associative array of attributes to add
- *string* **$noFrames**: The message to display if `iframes` are not supported
- *string* : Returns formatted XHTML string

image

This method generates the HTML to display a `<img></img>` element.

string **image($url**, **$alt**, [**$attribs** = null])

- *string* **$url**: The relative or absolute URL to use for the `src` attribute
- *string* **$alt**: The target attribute to use
- *array* **$attribs**: An associative array of attributes to add
- *string*: Returns formatted XHTML string

link

This method generates the HTML to display a `<a></a>` element.

string **link($url**, **$alt**, [**$attribs** = null])

- *string* **$url**: The relative URL to use for the `href` attribute
- *string* **$text**: The target attribute to use
- *array* **$attribs**: An associative array of attributes to add
- *string*: Returns formatted XHTML string

script

This method generates the HTML to display a `<script></script>` element.

string **script($filename**, [**$path** = 'media/system/js/'], [**$mootools** = true])

- *string* **$filename**: The name of the script file
- *string* **$path**: The relative or absolute path of the script file
- *boolean* **$mootools**: If `true` the mootools library will be loaded
- *string*: Returns formatted XHTML string

stylesheet

This method generates a `<link rel="stylesheet" style="text/css" />` element.

string **stylesheet($filename**, [**$path** = 'media/system/css/'], [**$attribs** = array()])

- *string* **$filename**: The relative URL to use for the `href` attribute
- *string* **$path**: The relative or absolute path of the stylesheet file
- *array* **$attribs**: An associative array of attributes to add
- *string*: Returns formatted XHTML string

tooltip

This method generates a tooltip with an image as a button.

string **tooltip**(**$tooltip**, [**$title** = ''], [**$image** = 'tooltip.png'], [**$text** = ''], [**$href** = ''], [**$link** = true])

- *string* **$tooltip**: The tip string
- *string* **$title**: Optional; the title of the tooltip
- *string* **$image**: The image for the tip if no tip text is provided
- *string* **$href**: Optional; a URL that will be used to create a link
- *boolean* **$link**: No longer used; deprecated
- *string*: Returns formatted XHTML string

Complex element types

The following classes provide group type methods and are described in detail in Chapter 8, *Rendering Output*. These group types are all invoked using the class loader method `JHTML::_()`.

`JHTMLBehavior`	Creates JavaScript client side behaviors
`JHTMLContent`	Fires `onPrepareContent` for non-article based content
`JHTMLEmail`	Provides email address cloaking
`JHTMLForm`	Generates hidden token field to reduce CSRF exploit risk
`JHTMLGrid`	Creates HTML form grids
`JHTMLImage`	Enables a type of image overriding in templates
`JHTMLList`	Generates common selection lists
`JHTMLMenu`	Used to generate menus
`JHTMLSelect`	Generate dropdown selection boxes

JHTMLBehavior

static, located in `/joomla/html/html/behavior.php`

This is a Joomla! utility class that generates element types that are special because they deal with JavaScript in order to create client-side behaviors. We call methods within this class through the class loader `JHTML::_()` method. This class makes extensive use of the `MooTools` JavaScript framework and specific JavaScript files located in the `/media/system/js/` folder.

Methods

calendar

This method adds the necessary JavaScript in order to use the JavaScript `showCalendar()` function to make date selection easier. If we want to use this when a user is not logged in we must add the `joomla.javascript.js` JavaScript file to the document:

```
$document =& JFactory::getDocument();
$document->addScript('includes/js/joomla.javascript.js');
```

Generally, we should use the basic `calendar` type instead.

void **calendar()**

- *void*: No return

caption

This method modifies images on a page that have a class of `caption` in such a way that the content of the image tag's title attribute appears beneath the image.

void **caption**()

- *void*: No return

combobox

This method adds JavaScript to modify the behavior of text fields (that are of class `combobox`) so as to add a combo selection. The available selections must be defined in an unordered list with the ID `combobox-idOfTheField`.

void **combobox()**

- *void*: No return

formvalidation

This method adds the generic `JFormValidator` JavaScript class to the document and instantiates an object of this type in `document.formvalidator`. This object can be used to aid in the validation of forms.

void **formvalidation()**

- *void*: No return

keepalive

This method adds a special invisible floating frame to the response that is updated regularly in order to maintain a user's session. This is of particular use with pages on which a user is likely to spend a long time creating or editing content.

void **keepalive()**

- *void*: No return

modal

This method adds JavaScript that enables us to implement modal windows. Modal windows are essentially inline popups that prevent the user from performing actions elsewhere on the page until the modal window has been closed. `$selector` determines which links should use modal windows; the default is `a.modal`.

void **modal**([**$selector** = 'a.modal'], [**$params** = array()])

- *string* **$selector**: Optional; which links should use modal windows
- *array* **$params**: An associative array of default modal window options
- *void*: No return

mootools

This method adds the `mootools` JavaScript library into the document head.

void **mootools**([**$debug** = null])

- *boolean* **$debug**: Optional; if `true` an uncompressed version is included
- *void*: No return

switcher

This method adds JavaScript that can be used to toggle between hidden and shown page elements. This is specifically used in conjunction with the backend sub-menu. For example, both the site configuration and system information areas in the backend use this.

void **switcher()**

- *void*: No return

tooltip

This method adds the necessary JavaScript to enable tooltips, the `mootools` JavaScript class `Tips`. To create tooltips we use the basic `tooltip` type, described earlier with the `JHTML` class. `$params` is an associative array of options; possible options include `maxTitleChars`, `timeout`, `showDelay`, `hideDelay`, `className`, `fixed`, `onShow`, and `onHide`.

void **tooltip**([**$selector** = 'hasTip'], [**$params** = array()])

- *string* **$selector**: Optional; class suffix, default is `hasTip`
- *array* **$params**: An associative array of options
- *void*: No return

tree

This method adds the necessary JavaScript to instantiate the `mootools` JavaScript class `MooTree`. `$params` is an optional associative array of global settings that may contain the keys `div`, `mode`, `grid`, `theme`, `loader`, `onExpand`, and `onSelect`. `$root` is an associative array that is used to create the root tree node. It may contain the keys `text`, `id`, `color`, `open`, `icon`, `openicon`, `data`, `onExpand`, and `onSelect`. The `text` key is the displayed text of the node and is required; all other keys are optional. The media manager uses this behavior. Further documentation can be found in `/media/system/js/mootree.js`.

void **tree**(**$id**, [**$params** = array()], [**$root** = array()])

- *string* **$id**: The unique id of the tree
- *array* **$params**: An associative array of options
- *array* **$root**: An associative array of `MooTreeNode` options for the root node
- *void*: No return

uploader

This method adds JavaScript that enables us to create a dynamic file uploading mechanism that allows users to upload a queue of files. The media manager uses this behavior.

void **uploader**([**$id** = 'file-upload'], [**$params** = array()])

- *string* **$id**: Optional; unique control id
- *array* **$params**: An associative array of options
- *void*: No return

JHTMLEMail

static, located in `/joomla/html/html/email.php`

This is a Joomla! utility class with only one method: `cloak`. We call the `cloak` method through the class loader `JHTML::_()` method.

Methods

cloak

This method uses JavaScript to display an encrypted email address in the browser. This prevents spam-bots, which crawl websites looking for email addresses, from discovering the email address. The form of encryption is very limited and is not a guaranteed way of beating spam-bots.

string **cloak**(**$mail**, [**$mailto** = true], [**$text** = ''], [**$email** = true],
[**$prefix** = 'mailto:'], [**$suffix** = ''], [**$attribs** = ''])

- *string* **$mail**: The un-encoded email address
- *boolean* **$mailto**: Optional; if `true` create mailto link, default is `true`
- *string* **$text**: Optional; alternative text to display
- *boolean* **$email**: Optional; if `true` (default), `$text` is an email address
- *string* **$prefix**: Optional; defaults to `'mailto:'`
- *string* **$suffix**: Optional; email address suffix, defaults to `''`
- *string* **$attribs**: Optional; additional email address attributes, defaults to `''`
- *string*: Returns a JavaScript string used to display encoded email address

JHTMLForm

static, located in `/joomla/html/html/form.php`

This is a Joomla! utility class with only one method: `token`. We call the `token` method through the class loader `JHTML::_()` method.

Methods

token

This method generates a hidden token field within a form to reduce the risk of CSRF exploits. It is used in conjunction with `JRequest::checkToken`.

string **token()**

- *string*: Returns an html hidden input element string

JHTMLGrid

static, located in `/joomla/html/html/grid.php`

This is a Joomla! utility class that is used to create HTML grids. There are seven grid types, each of which handles a common field found in the database. We call the methods through the class loader `JHTML::_()` method.

Methods

access

This method generates a text link that describes the access group (legacy group) to which the item is subject. When pressed the access of the item is designed to cycle through the available legacy groups and submits the form with the task `accessregistered`, `accessspecial`, or `accesspublic`. The referenced object `&$row` must contain the attributes `access` and `groupname`.

string **access**(*object* **&$row**, *integer* **$i**, [*integer* **$archived** = null])

- *object* **$row**: A reference to the represented object
- *integer* **$i**: The physical row number
- *integer* **$archived**: Optional; if `-1` the item is archived
- *string*: Returns a text link that describes the access group

checkedOut

This method generates a selectable checkbox or displays a small padlock image if the record is locked. The referenced object `&$row` must contain the attribute `checked_out` or be a `JTable` object. The returned checkbox is a member of a checkbox array whose value is equal to the record ID value. If the row/record is checked out a small padlock image is returned.

string **checkedOut**(*object* **&$row**, *integer* **$i**, [*string* **$identifier** = 'id'])

- *object* **$row**: A reference to the represented object
- *integer* **$i**: The physical row number
- *string* **$identifier**: Optional; primary key name, default is `id`
- *string*: Returns a selectable checkbox or small padlock image

id

This method generates a selectable checkbox. If `checkedOut` is true a null string is returned. This method is used by most of the other `grid` types; it is recommended that all admin grids/tables use this method. If the record might be checked out we should consider using `grid.checkedOut` instead. The returned checkbox is a member of a checkbox array whose value is equal to the `$recId`.

string **id**(**$rowNum**, **$recId**, [**$checkedOut** = false], [**$name** = 'cid'])

- *integer* **$rowNum**: The physical row number
- *integer* **$recId**: The record id
- *boolean* **$checkedOut**: Optional; if `false` (default) the record is checked in
- *string*: Returns a selectable checkbox

order

This method outputs an image with an `onClick` JavaScript to be used at the top of an `order` column. Every data row cell in this column will normally contain a text box called `order`, as this example demonstrates:

```
<input type="text" name="order[]" size="5"
       value="<?php echo $row->ordering;?>"
       class="text_area" style="text-align: center" />
```

string **order**(**$rows**, [**$image** = 'filesave.png'] , [**$task** = 'saveorder'])

- *array* **$rows**: An array of rows being displayed
- *string* **$image**: Optional; the image name, default is `'filesave.png'`
- *string* **$task**: Optional; the update order task; default is `'saveorder'`
- *string*: Returns a selectable checkbox or small disc image

published

This method outputs an image that represents a published state. When pressed the image issues a JavaScript event selecting the item, submitting the form with the task `publish` or `unpublish`. `$imgY` and `$imgX` default images are located in `/images`.

string **published**(**&$row**, **$i**, [**$imgY** = 'tick.png'],
[**$imgX** = 'publish_x.png'], [**$prefix** = ''])

- *object* **$row**: A reference to the represented object
- *integer* **$i**: The physical row number
- *string* **$imgY**: Optional; published image name
- *string* **$imgX**: Optional; unpublished image name
- *string* **$prefix**: Optional; task name prefix
- *string*: Returns an image used to publish and unpublish an item

sort

This method outputs a sortable heading for a grid/table column. When pressed sets the form fields `filter_order` and `filter_order_Dir` to the current column and the preferred direction. `$direction` contains the current direction, `filter_order_Dir` is populated with the opposite, either `asc` or `desc`.

string **sort**(**$title**, *integer* **$order**, [**$direction** = 'asc'], [**$selected** = 0], [**$task** = null])

- *string* **$title**: The column name
- *integer* **$order**: Value with which to populate `filter_order`
- *string* **$direction**: Optional; current direction
- *integer* **$selected**: Optional; the currently selected ordering column
- *string* **$task**: Optional; value to populate `task`
- *string*: Returns a sortable heading for a grid or table column

state

This method outputs a drop-down selection box called `filter_state` with four or five options. It is normally used to select the `published`, `unpublished`, or `archived` state. When an option is selected the form is submitted. The `$filter_state` must be either a null string, `'*'`, `'P'`, `'U'`, `'A'` or `'T'`. The default value for `$archived` is null, which prevents the archived option from being displayed. The default value for `$trashed` is null, which prevents the trashed option from being displayed.

string **state**([**$filter_state** = '*'], [**$published** = 'Published'],
[**$unpublished** = 'Unpublished'], [**$archived** = null], [**$trashed** = null])

- *string* **$filter_state**: Optional; the current state, default is `'*'`
- *string* **$published**: Optional; published name, default `'Published'`
- *string* **$unpublished**: Optional; unpublished name, default `'Unpublished'`
- *integer* **$archived**: Optional; archived name, default is null
- *string* **$trashed**: Optional; trashed name, default is null
- *string*: Returns a drop-down list of available states

JHTMLImage

static, located in `/joomla/html/html/image.php`

This is a Joomla! utility class that enables a form of image overriding by checking whether a template has an image before using a system default image. There are two image types, `administrator` and `site`. We call the class methods through the class loader `JHTML::_()` method.

Methods

administrator

This method determines if an image exists in the current administrator (backend) templates image directory. If it does it loads this image; otherwise, the default image is loaded. This method can also be used in conjunction with the `menulist` parameter to create the chosen image. If `$param` is null the `$file` and `$directory` values will be used, if `$param = -1` no image will be shown, and if `$param` contains a string value it will be used as the alternate image file name with `$param_directory`, the alternate path.

string **administrator**(**$file**, [**$directory** = '/images/'], [**$param** = null], [**$param_directory** = '/images/'], [**$alt** = null], [**$attribs** = null], [**$type** = true])

- *string* **$file**: The name of the image file
- *string* **$directory**: Optional; image directory, default `'/images/'`
- *mixed* **$param**: Optional; alternate image file name
- *integer* **$param_directory**: Optional; alternate path, default `'/images/'`
- *string* **$alt**: Optional; alt text, default is null
- *array* **$attribs**: Optional; an associative array of attributes, default is null
- *boolean* **$type**: Optional; if `true` display full tag, `false` just the path
- *string*: Returns an html image tag or image path string

site

This method determines if an image exists in the current frontend templates image directory. If it does it loads this image; otherwise, the default image is loaded. This method can also be used in conjunction with the `menulist` parameter to create the chosen image. If `$altFile` is null the `$file` and `$folder` values will be used, if `$altFile = -1` no image will be shown, and if `$altFile` contains a string value it will be used as the alternate image file name with `$altFolder`, the alternate path.

string **site**(**$file**, [**$folder** = '/images/M_images/'], [**$altFile** = null], [**$altFolder** = '/images/M_images/'], [**$alt** = null], [**$attribs** = null], [**$asTag** = true])

- *string* **$file**: The name of the image file
- *string* **$folder**: Optional; image directory, default `'/images/M_images/'`
- *mixed* **$altFile**: Optional; alternate image file name
- *integer* **$altFolder**: Optional; alternate path, default `'/images/M_images/'`
- *string* **$alt**: Optional; alt text, default is null
- *array* **$attribs**: Optional; an associative array of attributes, default is null
- *boolean* **$asTag**: Optional; if `true` display full tag, `false` just the path
- *string*: Returns an html image tag or image path string

JHTMLList

static, located in `/joomla/html/html/list.php`

This is a Joomla! utility class used to generate common selection lists that are generally used to implement a filter when viewing itemized data or when creating or modifying a single item. We call the class methods through the class loader `JHTML::_()` method.

Methods

accesslevel

This method generates a drop-down selection box of access level groups. The selected group will be the group identified in the `$row` attribute `access`. The resulting form control is named `access`.

string **accesslevel**(**&$row**)

- *object* **$row**: A reference to the row object
- *string*: Returns an HTML drop-down selection box of access level groups

category

This method generates a drop-down selection box of different categories related to a specific section. We can use categories outside of the content component in order to maintain categories for a different extension. We do this by specifying a section value equal to the extension name, for example `com_somecomponent`. If `$sel_cat` is `true` (default) display `'Select a Category'` at the top of the category list.

string **category**(**$name**, **$section**, [**$active** = null], [**$javascript** = null], [**$order** = 'ordering'], [**$size** = 1], [**$sel_cat** = true])

- *string* **$name**: The name of the list
- *string* **$section**: The section ID or extension name
- *string* **$active**: Optional; key value of initial selected category
- *string* **$javascript**: Optional; JavaScript event attributes to add to select tag
- *string* **$order**: Optional; SQL `ORDER BY` clause, default is `'ordering'`
- *integer* **$size**: Optional; Selection box size, default is 1
- *boolean* **$sel_cat**: Optional; if `true` display `'Select a Category'` at top
- *string*: Returns an HTML drop-down selection box string

genericordering

This method generates an array of `option` `stdClass` objects with properties of `value` and `text`. The array is used with the `select` types. If the current position is known consider using `grid.specificordering`. If there are no rows returned the method returns `false`, otherwise it returns a single `stdClass` object or an array of `stdClass` objects.

mixed **genericorderingt**(**$sql**, [**$chop** = '30'])

- *object* **$sql**: A SQL query to execute; must return the fields `text` and `value`
- *integer* **$chop**: Optional; Maximum text length, default is `30` characters
- *mixed*: Returns `false`, a `stdClass` object, or an array of `stdClass` objects

images

This method generates a drop-down list of images available in a directory. The first option in the list is always `Select Image`. Images must be of type `BMP`, `GIF`, `JPG`, or `PNG`. By default the list has a JavaScript `onChange` event associated with it that will update the `src` attribute of an `img` tag called `imagelib`.

string **images**(**$name**, [**$active** = null], [**$javascript** = null], [**$directory** = null],
[**$extensions** = 'bmp | gif | jpg | png'])

- *string* **$name**: The name of the list
- *string* **$active**: Optional; key value of initially selected image
- *string* **$javascript**: Optional; JavaScript event attributes to add
- *string* **$directory**: Optional; images directory, default `'/images/stories'`
- *string* **$extensions**: Optional; valid extension list
- *string*: Returns an HTML drop-down list with image names

positions

This method generates a drop-down list of positions; this is intended to enable the selection of image positions but can be used for other purposes. Valid positions are none, center, left, and right.

string **positions**(**$name**, [**$active** = null], [**$javascript** = null], [**$none** = true],
[**$center** = true], [**$left** = true], [**$right** = true], [**$id** = false])

- *string* **$name**: The name of the drop-down list form control
- *string* **$active**: Optional; key value of initially selected position
- *string* **$javascript**: Optional; JavaScript event attributes to add
- *boolean* **$none**: Optional; if `true` display `'None'`
- *boolean* **$center**: Optional; if `true` display `'Center'`
- *boolean* **$left**: Optional; if `true` display `'Left'`
- *boolean* **$right**: Optional; if `true` display `'Right'`
- *string* **$id**: Optional; drop-down list id
- *string*: Returns an HTML drop-down list of positions

section

This method generates a drop-down list of published sections within the specified scope. The first option is always `'- Select Section -'`. If `$uncategorized` is `true` the second option will be `'- Uncategorized -'`; the default is `true`. `$order` contains the SQL `ORDER BY` clause used when selecting sections from the `#__sections` table; the default is `'ordering'`.

string **section(\$name**, [**\$active** = null], [**\$javascript** = null], [**\$order** = 'ordering'], [**\$uncategorized** = true], [**\$scope** = 'content'])

- *string* **\$name**: The name of the drop-down list form control
- *string* **\$active**: Optional; key value of initially selected section
- *string* **\$javascript**: Optional; JavaScript event attributes to add
- *string* **\$order**: Optional; SQL `ORDER BY` clause
- *boolean* **\$uncategorized**: Optional; if `true` display `'- Uncategorized -'`
- *string* **\$scope**: Optional; the section scope; default is `'content'`
- *string*: Returns an HTML drop-down list of sections

specificordering

This method generates a drop-down list of possible positions in an order. `$row` is a reference to an object which represents the current item. If `$id` is `false`, a hidden input field is returned concatenated with a text message (for example the `com_weblinks` component displays the descriptive message when creating the first Web Link: *New Web Links default to the last position. Ordering can be changed after this Web Link is saved*. The description is related to creating new items; we use `$neworder` to suggest that the item will be placed at the start or end of the existing order. The returned control is named `ordering`. The first option item will be `'0 First'` and the last will be `'n Last'` where `n` will be 1 greater than the number of rows in the list.

string **specificordering**(**&\$row, \$id, \$query**, [**\$neworder** = false])

- *string* **\$row**: The name of the drop-down list form control
- *boolean* **\$id**: Optional; key value of initially selected section
- *string* **\$query**: Optional; JavaScript event attributes to add
- *boolean* **\$neworder**: Optional; if `true` new item will be first, `false` last
- *string*: Returns an html drop-down list of sections

users

This method generates a drop-down list of site users. If `$nouser` is `true` then the first item in the list will be `'- No User -'`, the default is `false`. If `$reg` is `true`, the default, then registered users will not be included in the list.

string **users**(**$name**, **$active**, [**$nouser** = false], [**$javascript** = null], [**$order** = 'name'], [**$reg** = false])

- *string* **$name**: The name of the drop-down list form control
- *integer* **$active**: The initially selected user id
- *boolean* **$nouser**: Optional; if true include `'- No User -'` option
- *string* **$order**: The #__users field to order by; default is `'name'`
- *boolean* **$reg**: Optional; if `true` registered users will not be listed
- *string*: Returns an html drop-down list of site users

JHTMLMenu

static, located in `/joomla/html/html/menu.php`

This is a Joomla! utility class that is used to work with menu select lists. It is unlikely that we should ever need to use any of these because menus are handled by Joomla! The `menu.treerecurse` type may be of interest when rendering tree structures.

Methods

linkoptions

This method generates an array of options for use with `select.genericlist` which represents the menu items.

array **linkoptions**([**$all** = false, [**$unassigned** = false])

- *boolean* **$all**: Optional; if `true` include `'All'` to the list
- *boolean* **$unassigned**: Optional; if `true` include `'Unassigned'` to the list
- *array*: Returns an array of options

ordering

This method generates a drop-down list of menu items from a menu in order to facilitate the modification of menu item ordering. The value of each option is equal to the ordering value of the corresponding menu item. If `$id` is `false`, a hidden field will be returned along with a textual description. The description explains that new items will be added to the end of the existing order.

string **ordering**(**&$row**, **$id**)

- *object* **$row**: A reference to the represented object
- *integer* **$i**: The physical row number
- *string*: Returns a drop-down list of menu items from a menu

treerecurse

This method builds an array of objects from menu items recursively. It adds the attributes `treename` and `children`. `treename` is the text to display before an item. `children` is the number of child menu items.

string **treerecurse**(**$id**, **$indent**, **$list**, **&$children**,
[**$maxlevel** = 9999], [**$level** = 0], [**$type** = true])

- *integer* **$id**: The ID of the menu item to build the array
- *integer* **$indent**: The current indentation level
- *array* **$list**: An array of menu items, normally empty on initial pass
- *array* **$children**: A reference to an array of objects representing menu items
- *integer* **$maxlevel**: Optional; maximum recursive depth, default is `9999`
- *integer* **$level**: Optional; the current recursive level, default is `0`
- *boolean* **$type**: Optional; if `true` pretext is `'|_'`, `false` is `'- '`
- *string*: Returns a selectable checkbox

JHTMLSelect

static, located in `/joomla/html/html/select.php`

This is a Joomla! utility class that is used to create drop-down selection boxes or radio selection buttons easily.

Methods

booleanlist

This method generates a pair of radio button options, one with a key of `0` and a value of `'yes'` and the other with a key of `1` and a value of `'no'`. The values can be changed using the `$yes` and `$no` parameters.

string **booleanlist**(**$name**, [**$attribs** = null], [**$selected** = null], [**$yes** = 'yes'], [**$no** = 'no'], [**$id** = false])

- *string* **$name**: The HTML `name` attribute
- *string* **$attribs**: Optional; additional HTML attributes for the `<select>` tag
- *mixed* **$selected**: The key that is selected
- *string* **$yes**: Optional; the `true` or `1` text, the default is `'yes'`
- *string* **$no**: Optional; the `false` or `0` text, the default is `'no'`
- *integer* **$id**: Optional; the button to be selected, default is `false`
- *string*: Returns HTML for a pair of radio buttons

genericlist

This method generates a drop-down selection list based on an array of options.

string **genericlist** (**$arr**, **$name**, [**$attribs** = null], [**$key** = 'value'], [**$text** = 'text'], [**$selected** = null], [**$idtag** = false], [**$translate** = false])

- *array* **$arr**: An array of arrays or objects, normally objects
- *string* **$name**: The value of the HTML `name` attribute
- *string* **$attribs**: Optional; additional HTML attributes for the `<select>` tag
- *string* **$key**: Optional; the value key in the associative array or objects
- *string* **$text**: Optional; the text key in the associative array or objects
- *mixed* **$selected**: Optional; the key value of the currently selected option
- *integer* **$idtag**: Optional; the list ID, default is `false`
- *boolean* **$translate**: Optional; if `true` translate the text using `JText`
- *string*: Returns a HTML drop-down select list with options

integerlist

This method generates a drop-down selection list of integers.

string **integerlist**(**$start**, **$end**, **$inc**, **$name**, [**$attribs** = null], [**$selected** = null], [**$format** = ''])

- *integer* **$start**: The start/minimum integer
- *integer* **$end**: The end/maximum integer
- *integer* **$inc**: The incremental value, normally 1
- *string* **$name**: The value of the HTML name attribute
- *string* **$attribs**: Optional; additional HTML attributes for the `<select>` tag
- *mixed* **$selected**: Optional; the currently selected value, default is null
- *string* **$format**: Optional; the `sprint()` format to apply to the text
- *string*: Returns a HTML drop-down select list of integers

optgroup

This method generates a `stdClass` object that represents an option group. The value property is set to `'<OPTGROUP>'` and the text property is set to `$text`.

object **optgroup**(**$text**, [**$value_name** = 'value'], [**$text_name** = 'text'])

- *string* **$text**: The value of the HTML name attribute
- *string* **$value_name**: Optional; the value property name, default is `'value'`
- *string* **$text_name**: Optional; the text property name, default is `'text'`
- *string*: Returns a `stdClass` object representing an option group

option

This method generates a `stdClass` object that represents a single selectable option with two properties, `$value_name` and `$text_name`. The `$obj->$value_name` property is set to `$value`; the `$obj->$text_name` property is set to `$text` if it is not empty; if `$text` is empty `$obj->$text_name` is set to `$value`.

object **option**(**$value**, [**$text** = ''], [**$value_name** = 'value'], [**$text_name** = 'text'], [**$disable** = false])

- *string* **$value**: The option value
- *string* **$text**: Optional; the option text
- *string* **$value_name**: Optional; the value property name, default is `'value'`
- *string* **$text_name**: Optional; the text property name, default is `'text'`
- *boolean* **$disable**: Optional; if `true` disable the option, default is `false`
- *string*: Returns a `stdClass` object representing a single selectable option

options

This method generates the option tags for a XHTML select list. `$arr` is an array of associative arrays or objects; it is normally an array of `stdClass` objects created by using `select.option()` and `select.optgroup()`.

string **options**(**$arr**, [**$key** = 'value'], [**$text** = 'text'],
[**$selected** = null], [**$translate** = false])

- *array* **$arr**: An array of associative arrays or objects
- *string* **$key**: Optional; the value property name, default is `'value'`
- *string* **$text**: Optional; the text property name, default is `'text'`
- *mixed* **$selected**: Optional; the key value of the currently selected option
- *boolean* **$translate**: Optional; if `true` translate the text using `JText`
- *string*: Returns a XHTML string of option tags

radiolist

This method generates a radio button selection list. `$arr` is an array of associative arrays or objects; it is normally an array of `stdClass` objects created by using `select.option()`.

string **radiolist**(**$arr**, **$name**, [**$attribs** = null], [**$key** = 'value'], [**$text** = 'text'],
[**$selected** = null], [**$idtag** = false], [**$translate** = false])

- *arr* **$arr**: An array of associative arrays or objects
- *string* **$name**: The value of the HTML `name` attribute
- *string* **$attribs**: Optional; additional HTML attributes for the `<select>` tag
- *string* **$key**: Optional; the value property name, default is `'value'`
- *string* **$text**: Optional; the text property name, default is `'text'`
- *integer* **$idtag**: Optional; the list ID, default is `false`
- *boolean* **$translate**: Optional; if `true` translate the text using `JText`
- *string*: Returns a radio button selection list

JPagination

extends `JObject`*, located in* `/joomla/html/pagination.php`

This class provides a common interface for content pagination. For further information on using the `JPagination` class refer to Chapter 8, *Rendering Output*.

Properties

integer `$limit` = null	The number of rows to display per page
integer `$limitstart` = null	The row number to start displaying from
integer `$total` = null	The total number of rows
boolean `$_viewall` = false	View all rows flag

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JPagination` object. Redefinition of `JObject::__construct()`; overridden in descendant classes.

JPagination **__construct(**$total**, **$limitstart**, **$limit**)**

- *integer* **$total**: The total number of items
- *integer* **$limitstart**: The offset of the item to start
- *integer* **$limit**: The number of items to display per page

getData

This method returns a `stdClass` data object that contains a set of `JPaginationObject` objects created to hold start, end, current, viewall, next, and previous pagination settings. If the data object does not exist it creates it.

object **getData()**

- *object*: Returns a `stdClass` data object of `JPaginationObject` objects

getLimitBox

This method creates a dropdown box for selecting how many records to show per page.

string **getLimitBox()**

- *string*: Returns the HTML for the limit input box

getListFooter

This method returns an HTML string that contains all of the elements that make up the pagination footer. The pagination footer is built from an associative array that contains elements with key names of `limit`, `limitstart`, `total`, `limitfield`, `pagescounter`, and `pageslinks`.

If there is a function `pagination_list_footer()` located in a `/html/pagination.php` file of the current template then that method will be executed, otherwise a private method `_list_footer()` will be executed. Either will return a formatted HTML string containing all of the HTML elements contained in the associative array that make up the pagination footer.

string **getListFooter()**

- *string*: Returns the pagination footer formatted as an HTML string

getPagesCounter

This method creates and returns the pagination pages counter string, for example: **Page 2 of 4**.

string **getPagesCounter()**

- *string*: Returns the HTML pagination pages counter

getPagesLinks

This method creates and returns the pagination page list string, for example: **Previous, Next, 1 2 3 ...x**.

If there is a function `pagination_list_render()` located in a `/html/pagination.php` file of the current template then that method will be executed, otherwise a private method `_list_render()` will be executed. Either will return a formatted HTML string containing the pagination list string.

string **getPageLinks**()

- *string*: Returns the HTML pagination page list string

getResultsCounter

This method creates and returns the pagination result set counter string,
for example:
Results 1-10 of 42.

string **getResultsCounter**()

- *string*: Returns the HTML pagination result set counter string

getRowOffset

This method returns the rationalized offset for a row with a given index (`$index + $limitstart + 1`). For example, if `$index=10` and `$limitstart=50` then the method will return `61` as the row index.

integer **getRowOffset**(**$index**)

- *integer* **$index**: The row index
- *integer*: Returns rationalized offset for a row with a given index

orderDownIcon

This method returns the HTML to display an icon to move an item down in the current ordering.

string **orderDownIcon(**$**i**, $**n**, [$**condition** = true**]**, [$**task** = 'orderdown'**]**, [$**alt** = 'Move Down'**]**, [$**enabled** = true**]**)

- *integer* **$i**: The row index
- *integer* **$n**: The number of items in the list
- *boolean* **$condition**: If `true` show the icon
- *string* **$task**: The task to fire
- *string* **$alt**: The image alternate text string
- *boolean* **$enabled**: If `true` return HTML to display icon; `false` return `&nbsp;`
- *string*: Returns HTML string to display icon to move an item down or `&nbsp;`

orderUpIcon

This method returns the HTML to display an icon to move an item up in the current ordering.

string **orderUpIcon(**$**i**, [$**condition** = true**]**, [$**task** = 'orderup'**]**, [$**alt** = 'Move Up'**]**, [$**enabled** = true**]**)

- *integer* **$i**: The row index
- *boolean* **$condition**: If `true` show the icon
- *string* **$task**: The task to fire
- *string* **$alt**: The image alternate text string
- *boolean* **$enabled**: If `true` return HTML to display icon; `false` return `&nbsp;`
- *string*: Returns HTML string to display the icon to move an item up or `&nbsp;`

_buildDataObject

This *private* method creates and returns the pagination `stdClass` data object.

object **_buildDataObject()**

- *object*: Returns a `stdClass` object containing `JPaginationObject` objects

_item_active

This *private* method generates the HTML link `<a></a>` tag to display the next set of rows.

string **_item_active(&$item)**

- *object* **$item**: The `JPaginationObject` object to build the link
- *string*: Returns the link to click for the next set of rows to display

_item_inactive

This *private* method generates the HTML `<span></span>` tag to display the text field of the `JPaginationObject` item. No link is generated.

string **_item_inactive(&$item)**

- *object* **$item**: The `JPaginationObject` object to build the HTML
- *string*: Returns the text from the `JPaginationObject` object in span tags

_list_footer

This *private* method generates the HTML string to display the pagination footer (See `getListFooter()`).

string **_list_footer($list)**

- *array* **$list**: The associative array of pagination footer elements
- *string*: Returns the HTML string to display the pagination footer

_list_render

This *private* method generates the HTML string to display the pagination page list (See `getPageLinks()`).

string **_list_footer($list)**

- *array* **$list**: The associative array of pagination page list elements
- *string*: Returns the HTML string to display the pagination page list

JPaginationObject

extends `JObject`, *located in* `/joomla/html/pagination.php`

This class is a pagination object representing a particular item in the pagination lists. For further information on using the `JPaginationObject` class refer to Chapter 8, *Rendering Output*.

Properties

integer `$base` = null	Normally the offset or next row
mixed `$link` = null	Link for URL (for example - `"&limitstart=".$page`)
string `$text`	Identifies `JPaginationObject` purpose (for example - Next, End)

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JPaginationObject` object. Redefinition of `JObject::__construct()`; overridden in descendant classes.

JPaginationObject **__construct($text**, **$base** = null, **$link** = null)

- *string* **$text**: Identifies the object's purpose (for example - Next, End)
- *integer* **$base**: The offset of the item to start
- *mixed* **$link**: Link for URL (for example - `"&limitstart=".$page`)

JPane

abstract, extends `JObject`, *located in* `/joomla/html/pane.php`

This is an abstract class that is the base class for JPaneTabs and JPaneSliders classes that use a mixture of XHTML and JavaScript behaviors used to create tabbed and slider panes. For more information about the `JPanes` class refer to Chapter 9, *Customizing the Page*.

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Redefined and overridden in descendant classes.

JPane **__construct**([**$params** = array()])

- *array* **$params**: Configuration settings

endPane

This *abstract* method ends the pane definition. Redefined in descendant classes.

void **endPane()**

- *void*: No return

endPanel

This *abstract* method ends the panel definition. Redefined in descendant classes.

void **endPanel()**

- *void*: No return

getInstance

This method returns a reference to a `JPane` object.

JPane **&getInstance([$behavior** = 'Tabs'], [**$params** = array()])

- *string* **$behavior**: The type of pane to create; either `'Tabs'` or `'Sliders'`
- *array* **$params**: Configuration settings
- *object*: Returns a reference to a JPane subclass object

startPane

This *abstract* method creates a pane. Redefined in descendant classes.

void **startPane($id)**

- *string* **$id**: The pane identifier
- *void*: No return

startPanel

This *abstract* method creates a panel with title text. Redefined in descendant classes.

void **startPanel($text, $id)**

- *string* **$text**: The panel name or title
- *string* **$id**: The panel identifier
- *void*: No return

_loadBehavior

This *abstract private* method loads the JavaScript behavior and attaches it to the document.

void **_loadBehavior()**

- *void*: No return

JPaneSliders

extends `JPane`, *located in* `/joomla/html/pane.php`

This class is used to create `JPaneSliders`. For more information about the `JPaneSliders` class refer to Chapter 9, *Customizing the Page*.

Inherited properties

Inherited from `JPane`:

- `JPane::$useCookies`

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JPane`:

- `JPane::__construct()`
- `JPane::endPane()`
- `JPane::endPanel()`
- `JPane::getInstance()`
- `JPane::startPane()`

- `JPane::startPanel()`
- `JPane::_loadBehavior()`

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JPaneSliders` object and loads the JavaScript behaviors, attaching them to the document.

JPaneSliders **__construct**([**$params** = array()])

- *array* **$params**: Configuration settings

endPane

This *abstract* method ends the pane definition with a closing `'</div>'` tag.

string **endPane()**

- *string*: Returns a `'</div>'` string.

endPanel

This method ends the panel definition with a closing `'</div></div>'` tag.

string **endPanel()**

- *string*: Returns a `'</div></div>'` string

startPane

This method creates a pane definition with:

```
'<div id="'.$id.'" class="pane-sliders">';
```

string **startPane($id)**

- *string* **$id**: The pane identifier
- *string*: Returns opening slider pane XHMTL string

startPanel

This method creates a panel with title text using:

```
  '<div class="panel">'
. '<h3 class="jpane-toggler title" id="'.$id.'">
       <span>'.$text.'</span></h3>'
. '<div class="jpane-slider content">';
```

void **startPanel($text, $id)**

- *string* **$text**: The panel name or title
- *string* **$id**: The panel identifier
- *void*: No return

_loadBehavior

This *private* method loads JavaScript behaviors and attaches them to the document.

void **_loadBehavior()**

- *void*: No return

JPaneTabs

extends `JPane`, *located in* `/joomla/html/pane.php`

This class is used to create `JPaneTabs`. For more information about the `JPaneTabs` class refer to Chapter 9, *Customizing the Page*.

Inherited properties

Inherited from `JPane`:

- `JPane::$useCookies`

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JPane`:

- `JPane::__construct()`
- `JPane::endPane()`
- `JPane::endPanel()`
- `JPane::getInstance()`
- `JPane::startPane()`
- `JPane::startPanel()`
- `JPane::_loadBehavior()`

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JPaneTabs` object and loads the JavaScript behaviors, attaching them to the document.

JPaneTabs **__construct**([**$params** = array()])

- *array* **$params**: Configuration settings

endPane

This *abstract* method ends the pane definition with a closing `'</dl>'` tag.

string **endPane()**

- *string* : Returns a `'</dl>'` string.

endPanel

This method ends the panel definition with a closing `'</dd>'` tag.

string **endPanel()**

- *string* : Returns a `'</dd>'` string.

startPane

This method creates a pane definition with:

```
'<dl class="tabs" id="'.$id.'">';
```

string **startPane($id**)

- *string* **$id**: The pane identifier
- *string*: Returns opening slider pane XHMTL string

startPanel

This method creates a panel with title text using:

```
'<dt id="'.$id.'"><span>'.$text.'</span></dt><dd>';
```

void **startPanel($text**, **$id**)

- *string* **$text**: The panel name or title
- *string* **$id**: The panel identifier
- *void*: No return

_loadBehavior

This *private* method loads JavaScript behaviors and attaches them to
the document.

void **_loadBehavior()**

- *void*: No return

F
Joomla! Utility Classes

Joomla! provides numerous utility classes to assist in performing many common tasks. This appendix provides details on those utility classes, including the following:

- JArchive
- JArrayHelper
- JDate
- JError
- JException
- JFile
- JFolder
- JFTP
- JHelp
- JLanguage
- JLanguageHelper
- JLDAP
- JLog
- JMail
- JMailHelper
- JNode
- JPath
- JSimpleXML
- JSimpleXMLElement
- JText
- JTree

JArchive

Static class, located in `/joomla/filesystem/archive.php`

This static class provides methods for creating and extracting archive files. Supported archive adapters include **bzip2**, **gzip**, **tar**, and **zip**. For more details see Chapter 12, *Utilities and Useful Classes*.

Methods

create

This method creates an archive file using the specified compression format.

Archive_Tar **create**(**$archive**, **$files**, [**$compress** = 'tar'] , [**$addPath** = ''], [**$removePath** = ''], [**$autoExt** = false], [**$cleanup** = false])

- *string* **$archive**: The name of the archive
- *mixed* **$files**: The name of a single file or an array of files
- *string* **$compress**: The compression format for the archive
- *string* **$addPath**: The path to add within the archive
- *string* **$removePath**: The path to remove within the archive
- *boolean* **$autoExt**: If true automatically append the extension
- *boolean* **$cleanup**: Remove the source files
- *object*: Returns an `Archive_Tar` object

extract

This method unpacks an archive file into the specified directory.

boolean **extract**(**$archivename**, **$extractdir**)

- *string* **$archivename**: The name of the archive file
- *string* **$extractdir**: The directory to unpack the archive file
- *boolean*: Returns `true` upon success

getAdapter

This method returns a reference to a `JArchive` adapter object.

JArchive **&getAdapter**(**$type**)

- *string* **$type**: The type of archive
- *object*: Returns a reference to a `JArchive` adapter object

JArrayHelper

Static class, located in `/joomla/utilities/arrayhelper.php`

This static class provides a number of useful methods to simplify common tasks when working with arrays. For more details see Chapter 12, *Utilities and Useful Classes*.

Methods

fromObject

This method maps an object to an array.

array **fromObject**(**$p_obj**, [**$recurse** = true], [**$regex** = null])

- *object* **$p_obj**: The source object
- *boolean* **$recurse**: If `true` recurse through multi-level objects
- *string* **$regex**: An optional regular expression to match on field names
- *array*: Returns an associative array mapped from the source object

getColumn

This method extracts a column from `$array`; `$array` may contain either objects or arrays.

array **getColumn**(**&$array**, **$index**)

- *array* **$array**: A reference to the source array
- *mixed* **$index**: The index of the column or name of the object property
- *array*: Returns an associative array representing a column of values

getValue

This method returns a value from a named array or a specified default. `$type` can have a value of `'INT'`, `'FLOAT'`, `'STRING'`, `'WORD'`, `'BOOLEAN'`, `'ARRAY'`.

mixed **getValue**(**&$array**, **$name**, [**$default** = null], [**$type** = ''])

- *array* **$array**: The named source array
- *string* **$name**: The search key
- *mixed* **$default**: Optional; the default value if no key is found
- *string* **$type**: Return type for the variable
- *mixed*: Returns a typed value

sortObjects

This method sorts an array of objects based on a specified field.

array **sortObjects**(**&$a**, **$k**, [**$direction** = 1])

- *array* **$a**: The source array of objects
- *string* **$k**: The key to sort on
- *integer* **$direction**: Optional direction to sort (1=ascending, -1=descending)
- *array*: Returns a sorted source array of objects

toInteger

This method converts the element values of an array to integer values. `$default` can be an array, a single value or null. If `$array` is an array each element is converted (cast) to an integer value. If `$array` is not an array its contents are determined by the value of `$default`.

- If `$default` is null, the value of `$array` is converted (cast) as an integer and then converted to an array
- If `$default` is an array, its elements are converted (cast) as integers and then assigned to `$array`
- If $`default` is not an array, its value is converted (cast) as an integer, converted to an array and assigned to `$array`

void **toInteger**(**&$array**, [**$default** = null])

- *array* **&$array**: The source array to convert
- *mixed* **$default**: The default value; can be a single value, an array, or null
- *array*: Returns the source array with values converted

toObject

This method creates a new `stdClass` object and adds properties to the object based on the array keys.

object **toObject**(**&$array**, [**$class** = 'stdClass'])

- *array* **&$array**: The source array to convert
- *string* **$class**: Optional name of the class to create
- *object*: Returns the source array as a `stdClass` object

toString

This method is a utility function that converts array values to a string. The source array can contain nested arrays which will also be converted. This method uses the PHP `implode` function.

string **toString**(**$array**, [**$inner_glue** = '='], [**$outer_glue** = ' '],
[**$keepOuterKey** = false])

- *array* **$array**: The source array to convert
- *string* **$inner_glue**: Connector string between key and value
- *string* **$outer_glue**: Connector string between key-value pairs
- *string*: Returns source array as a string

_sortObjects

This is a private callback method for sorting an array of objects based on a key. It is called by `JArrayHelper::sortObjects()`.

integer **_sortObjects**(**&$a**, **&$b**)

- *array* **&$a**: An array of objects
- *array* **&$b**: An array of objects
- *integer*: Returns an integer representing the direction of the sort

JDate

extends `JObject`, *located in* `/joomla/utilities/date.php`

This is the Joomla! date class that provides a number of useful methods for handling date and time values across different time zones and using different formats. For more information about the `JDate` class refer to Chapter 12, *Utilities and Useful Classes*.

Properties

mixed `$_date` = false	Unix timestamp or Boolean
integer `$_offset` = 0	Time offset in seconds

Inherited properties

Inherited from `JObject`:

- `JObject::_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor that creates a new instance of `JDate` representing a given date.

Redefines `JObject::__construct()`; overridden in descendant classes.

JDate **__construct**([**$date** = 'now'], [**$tzOffset** = 0])

- *mixed* **$date**: Optional; the date this `JDate` object will represent
- *integer* **$tzOffset**: Optional; the time zone of `$date`
- *object*: Returns a new `JDate` object

getOffset

This method returns the date offset (in hours).

integer **getOffset**()

- *integer*: Returns the date offset in hours

setOffset

This method sets the date offset (in hours).

void **setOffset**(**$offset**)

- *float* **$offset**: The offset in hours
- *void*: No return

toFormat

This method returns a string formatted according to the `$format`. Month and weekday names and other language dependent strings respect the current locale. The date format specification can be reviewed at `http://www.php.net/strftime`.

string **toFormat**([**$format** = '%Y-%m-%d %H:%M:%S'])

- *string* **$format**: The date format specification string
- *string*: Returns the formatted date

toISO8601

This method returns a date string formatted in ISO 8601 (RFC 3339) format.

The RFC specification can be reviewed at `http://www.ietf.org/rfc/rfc3339.txt?number=3339 (IETF RFC 333`9).

string **toISO8601**([**$local** = false])

- *boolean* **$local**: If `true` adjust the date by the time zone offset
- *string*: Returns the formatted date

toMySQL

This method returns a date string formatted in MySQL datetime format.

You can review the specification at `http://dev.mysql.com/doc/refman/4.1/en/datetime.html`.

string **toMySQL**([**$local** = false])

- *boolean* **$local**: If `true` adjust the date by the time zone offset
- *string*: Returns the formatted date

toRFC822

This method returns a date string formatted in RFC 822 format.

RFC 822 has been replaced with RFC 2822. The RFC specification can be reviewed at `http://www.ietf.org/rfc/rfc2822.txt?number=2822 (IETF RFC 2822).`

string **toRFC822**([**$local** = false])

- *boolean* **$local**: If `true` adjust the date by the time zone offset
- *string*: Returns the formatted date

toUnix

This *private* method returns a date string formatted as a UNIX time stamp.

string **toUnix**([**$local** = false])

- *boolean* **$local**: If `true` adjust the date by the time zone offset
- *string*: Returns the formatted date

_dayToString

This *private* method translates the weekday number to a weekday name. The first day of the week is Sunday and the number is zero-based, so 0 = Sunday, 6 = Saturday.

string **_dayToString**(**$day**, [**$abbr** = false])

- *integer* **$day**: The numeric day of the week
- *boolean* **$abbr**: If `true` return the abbreviated day of the week (for example SUN)
- *string*: Returns the day of the week as a string

_monthToString

This *private* method translates the number of the month to a month name. The first month is January and the month number begins with one so 1 = January, 12 = December.

string **_monthToString($month**, [**$abbr** = false])

- *integer* **$month**: The numeric month
- *boolean* **$abbr**: If `true` return the abbreviated month name (for example JAN)
- *string*: Returns the month number as a string

_strftime

This *private* method replaces parts of the date number with strings based on the format specified in `$format`. For example if `$format` contains `'%A, %d %B %Y'` then the method will return the string `'Thursday, 01 January 1970'`. The date format specification can be reviewed at `http://www.php.net/strftime`.

string **_strftime**(**$format**, **$time**)

- *string* **$format**: The date format specification string
- *integer* **$time**: Unix timestamp
- *string*: Returns a date in the specified format

JError

static, located in `/joomla/error/error.php`

This is the static error handling class. For more information about `JError` refer to Chapter 11, *Error Handling and Security*.

Methods

attachHandler

This *static* method attaches the error handler and a custom error handler to `JError`.

void **attachHandler**()

- *void*: No return

customErrorHandler

This *static* method raises a custom error.

void **customErrorHandler**(**$level**, **$msg**)

- *void*: No return

customErrorPage

This *static* method gets the global instance of `JDocumentError` and passes it the error, then renders the page. This removes any previous output and terminates the script.

void **customErrorPage**(**&$error**)

- object **$error**: A reference to the `JException` error object
- *void*: No return

detachHandler

This *static* method detaches the error handler and the custom error handler from `JError`.

void **detachHandler**()

- *void*: No return

getError

This *static* method returns the last `JException` object on the global error stack or `false` if there are no errors. If `$unset` is `true` the error will be removed from the stack.

mixed **getError**(**$unset** = false)

- boolean **$unset**: If `true` removes the error from the stack
- *mixed*: Returns the last `JException` object or `false` if no errors

getErrorHandling

This *static* method returns a copy of the associative array used to determine the handling of the specified error level. The error level can be any of PHP's error levels (for example `E_ALL`, `E_NOTICE`, and so on) The array contains the key `mode` and, optionally, the key `options`. If an unknown level is passed, then `null` will be returned.

array **getErrorHandling**(**$level**)

- integer **$level**: The error level to retrieve
- *array*: Returns array of error handler settings for specified error level

getErrors

This *static* method returns a chronological array of errors that have been stored on the global `JError` exception stack during script execution.

array **&getErrors**()

- *array*: Returns an array of errors

handleCallback

This *static* method sends the `JError` object to a callback method for error handling. The `$options` array contains two elements, '`classname`' and '`methodname`', that will be the class and method to be executed.

JException **&handleCallback**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array with class and method to execute
- *object*: Returns the `JException` object

handleDie

This *static* method terminates the script and outputs the `JException` message to the screen in HTML. If `$_SERVER['HTTP_HOST']` is not set, the message will be echoed in plain text or, if `STDERR` is defined, written to `STDERR`.

JException **&handleDie**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array of handler options (ignored)
- *object*: Returns a reference to the `JException` object

handleEcho

This *static* method outputs the `JException` message to the screen in HTML. If `$_SERVER['HTTP_HOST']` is not set, the message will be echoed in plain text or, if `STDERR` is defined, written to `STDERR`.

JException **&handleEcho**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array of handler options (ignored)
- *object*: Returns a reference to the `JException` object

handleIgnore

This *static* method is the ignore error handler. No action is taken; a reference to the `JException` object is returned.

JException **&handleIgnore**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array of handler options (ignored)
- *object*: Returns a reference to the `JException` object

handleLog

This *static* method adds a log entry to the error log. A new error log is created every day in the format `Y-m-d.error.log`. The entry includes the date, time, level, code, and message.

JException **&handleLog**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array of handler options (passed to `JLog`)
- *object*: Returns a reference to the `JException` object

handleMessage

This *static* method triggers a PHP user-level error, warning, or notice.

JException **&handleMessage**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array of handler options (ignored)
- *object*: Returns a reference to the `JException` object

handleTrigger

This *static* method adds the `JException` message to the application message queue.

JException **&handleTrigger**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array of handler options (ignored)
- *object*: Returns a reference to the `JException` object

handleVerbose

This *static* method outputs a `JException` message, info, and backtrace to screen in HTML. If `$_SERVER['HTTP_HOST']` is not set, the message will be echoed in plain text or, if `STDERR` is defined, written to `STDERR`.

JException **&handleVerbose**(**&$error**, **$options** = array())

- *object* **$error**: The `JException` object to handle
- *array* **$options**: An array of handler options (ignored)
- *object*: Returns a reference to the `JException` object

isError

This *static* method determines whether a value is an exception object. This method supports both `JException` and PHP5 `Exception` objects.

boolean **isError**(**&$object**)

- *mixed* **$object**: The object to check
- *boolean*: Returns `true` if the argument is an exception, `false` otherwise

raise

This *static* method raises a new error of `$level` and executes the associated error handling mechanisms. `$level` relates to the PHP error levels, `E_NOTICE`, `E_WARNING`, and `E_ERROR`. Error handling levels and mechanisms can be altered. This method normally returns a reference to a `JException` object.

mixed **&raise**(**$level**, **$code**, **$msg**, [**$info** = null], [**$backtrace** = false])

- *integer* **$level**: The error level; use any of PHP's error levels
- *string* **$code**: The application-internal error code for this error
- *string* **$msg**: The error message that may be shown to the user
- *mixed* **$info**: Optional; additional error information (for developer use)
- *boolean* **$backtrace**: If `true` include `debug_backtrace()` information
- *mixed*: Returns a reference to a PHP exception or `JException` object

raiseError

This is a *static* wrapper method for the `raise()` method with a predefined error level of `E_ERROR` and `$backtrace` set to `true`; it executes the associated error handling mechanisms (by default `JError::handleCallback()`, which in turn calls `JError::customErrorPage()`).

object **&raiseError**(**$code**, **$msg**, [**$info** = null])

- *string* **$code**: The application-internal error code for this error
- *string* **$msg**: The error message that may be shown to the user
- *mixed* **$info**: Optional; additional error information (for developer use)
- *object*: Returns a reference to a configured `JError` object

raiseNotice

This is a *static* wrapper method for the `raise()` method with a predefined error level of `E_NOTICE` and `$backtrace` set to `false`; it executes the associated error handling mechanisms (by default `JError::handleMessage`). Normally this method will display a notice message on the resultant page.

object **&raiseNotice**(**$code**, **$msg**, [**$info** = null])

- *string* **$code**: The application-internal error code for this error
- *string* **$msg**: The error message that may be shown to the user
- *mixed* **$info**: Optional; additional error information (for developer use)
- *object*: Returns a reference to a configured `JError` object

raiseWarning

This is a *static* wrapper method for the `raise()` method with a predefined error level of `E_WARNING` and `$backtrace` set to `false`; it executes the associated error handling mechanisms (by default `JError::handleMessage`). Normally this method will display a notice message on the resultant page.

object **&raiseWarning(\$code**, **\$msg**, [**\$info** = null])

- *string* **\$code**: The application-internal error code for this error
- *string* **\$msg**: The error message that may be shown to the user
- *mixed* **\$info**: Optional; additional error information (for developer use)
- *object*: Returns a reference to a configured `JError` object

registerErrorLevel

This *static* method registers a new error level for handling errors allowing you to add custom error levels to the built-in `E_ERROR`, `E_WARNING`, and `E_NOTICE` levels. If the level already exists it will be rejected. `$name` describes to the error type. `$handler` defines the mode to use when an error of the new level is encountered (`ignore`, `echo`, `verbose`, `die`, `messages`, or `log`). To use `callback`, use `JError::setErrorHandling()` after registering the new level.

boolean **registerErrorLevel(\$level**, **\$name**, [**\$handler** = 'ignore'])

- *integer* **\$level**: The error level to register
- *string* **\$name**: The human readable name for the error level
- *string* **\$handler**: Optional; error handler to set for the new error level
- *boolean*: Returns `true` upon success

setErrorHandling

This *static* method sets the error handling mechanism for `$level`. Only levels that have already been defined can be modified. `$mode` specifies what will occur when an error of the specified level is encountered. Error handling modes include:

- `ignore`
- `echo`
- `verbose`
- `die`
- `message`
- `log`
- `callback`

You may also set the error handling for several modes at once using PHP's bit operations. Examples include:

- `E_ALL` = Set the handling for all levels
- `E_ERROR | E_WARNING` = Set the handling for errors and warnings
- `E_ALL ^ E_ERROR` = Set the handling for all levels except errors

If `$options` is set it is passed to the handler method. For example, if `$mode` was `'message'`, then the `JError::handleMessage()` method would be called with two parameters—the `JException` object and `$options`. If `'mode'` is `callback`, the options array must be specified, and it must contain two string elements, a class name and a method to execute. `callback` is special, because it is the only mode in which a method outside of the `JError` class can be used to handle an error.

mixed **setErrorHandling**(**$level**, **$mode**, [**$options** = null])

- *integer* **$level**: The error level to set
- *string* **$mode**: The error handler
- *string* **$handler**: Optional; Options to pass to the handler method
- *boolean*: Returns `true` upon success, `JException` object if failure

translateErrorLevel

This *static* method translates an error level integer into a human readable string (for example, `E_ERROR` will be translated to 'Error'). If the error level is not defined, `false` will be returned.

mixed **translateErrorLevel**(**$level**)

- *integer* **$level**: The error level to translate
- *mixed*: Returns the translated error level name, `false` if undefined

JException

extends `JObject`, *located in* `/joomla/error/exception.php`

This class encapsulates error information. Whenever an error is raised in Joomla! a `JException` object is created that includes valuable information that will point to the source of the problem.

Properties

array `$args`	Arguments received by method where error occurred
mixed `$backtrace` = null	Backtrace information
string `$class` = null	Name of the class where the error occurred
string `$code` = null	Error code
string `$file` = null	Name of the file where the error occurred
string `$function` = null	Name of the method where the error occurred
string `$info` = ''	Additional info about the error relevant to the developer
string `$level` = null	Error level
integer `$line` = 0	Line number where the error occurred
string `$message` = null	Error message
string `$type` = null	Error type

Inherited properties

Inherited from `JObject`:

- `JObject::_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JException` object used to set up the error with all needed error details.

Redefines `JObject::__construct()`; overridden in descendant classes.

JException **__construct($msg**, **$code**, [**$level** = null],
[**$info** = null], [**$backtrace** = false])

- *string* **$msg**: The error message
- *string* **$code**: The error code from the application
- *integer* **$level**: The error level
- *string* **$info**: Optional; additional error information
- *string* **$backtrace**: If `true` collect backtrace information
- *object*: Returns a `JException` object

getCode

This method returns the exception code.

integer **getCode**()

- *integer*: Returns the exception code

getFile

This method returns the source file name where the exception occurred.

string **getFile**()

- *string*: Returns the source file name

getLine

This method returns the source line number where the exception occurred.

integer **getLine**()

- *integer*: Returns the source line number

getMessage

This method returns the exception message.

string **getMessage**()

- *string*: Returns the exception message

getTrace

This method returns the backtrace array.

array **getTrace**()

- *array*: Returns the backtrace array

getTraceAsString

This method returns the formatted backtrace information.

string **getTraceAsString**()

- *string*: Returns a formatted string of the trace

toString

This method returns the error message. Redefines `JObject::toString()`.

string **toString**()

- *string*: Returns the error message

JFile

static, located in `/joomla/filesystem/file.php`

This is the Joomla! static class for handling files. For more information about file handling see Chapter 12, *Utilities and Useful Classes*.

Methods

copy

This method copies a file to a new location.

boolean **copy**(**$src**, **$dest**, [**$path** = null])

- *string* **$src**: The full path (including file name) for the source file
- *string* **$dest**: The full path (including file name) to the destination file
- *string* **$path**: An optional base path to prefix to the file names
- *boolean*: Returns `true` upon success

delete

This method deletes a file or an array of files.

boolean **delete**(**$file**)

- *mixed* **$file**: The file name or an array of file names to be deleted
- *boolean*: Returns `true` upon success

exists

This method determines if a file exists.

boolean **exists**(**$file**)

- *string* **$file**: The full path (including file name)
- *boolean*: Returns `true` upon success

getExt

This method gets the extension of a file. The extension is determined by the position of the first dot (.) found in the file name. If a file name contains more than one dot, `file.name.ext` for example, then the extension returned will be `name.ext`.

string **getExt($file)**

- *string* **$file**: The file name
- *string*: Returns the file extension

getName

This method gets the name of the file without the path.

string **getName($file)**

- *string* **$file**: The full path (including file name)
- *string*: Returns the file name

makeSafe

This method sanitizes the file name and makes it safe to use. The method removes any characters that are unsafe to use in a file name. The `$file` parameter must not be a path to a file as the directory separators will be stripped.

string **makeSafe($file)**

- *string* **$file**: The name of the file (not the full path)
- *string*: Returns the sanitized file name

move

This method moves a file to a new location.

boolean **move($src**, **$dest**, [**$path** = null])

- *string* **$src**: The full path (including file name) of the source file
- *string* **$dest**: The full path (including file name) of the destination file
- *string* **$path**: An optional base path to prefix to the file names
- *boolean*: Returns `true` upon success

read

This method reads the contents of a file.

mixed **read**(**$filename**, [**$incpath** = false], [**$amount** = 0],
[**$chunksize** = 8192], [**$offset** = 0])

- *string* **$filename**: The full path (including file name)
- *boolean* **$incpath**: Use include path
- *integer* **$amount**: The amount of the file to read
- *integer* **$chunksize**: The size of the chunks to read
- *integer* **$offset**: The offset of the file
- *mixed*: Returns the file contents or `false` upon failure

stripExt

This method strips the last extension off a file name.

string **stripExt**(**$file**)

- *string* **$file**: The file name
- *string*: Returns the file name without the extension

upload

This method moves an uploaded file to a destination folder.

boolean **upload**(**$src**, **$dest**)

- *string* **$src**: The name of the PHP (temporary) uploaded file
- *string* **$dest**: The path (including file name) to move the uploaded file
- *boolean*: Returns `true` upon success

write

This method writes content to a file.

boolean **write**(**$file**, **$buffer**)

- *string* **$file**: The full path (including file name)
- *string* **$buffer**: The buffer to write
- *boolean*: Returns `true` upon success

JFolder

static, located in `/joomla/filesystem/folder.php`

This is the Joomla! static class for handling folders or directories. For more information about folders see Chapter 12, *Utilities and Useful Classes*.

Methods

copy

This method copies a folder to a new location.

mixed **copy**(**$src**, **$dest**, [**$path** = ''], [**$force** = false])

- *string* **$src**: The path to the source folder
- *string* **$dest**: The path to the destination folder
- *string* **$path**: An optional base path to prefix to the file names
- *boolean* **$force**: Optionally force folder/file overwrites
- *mixed*: Returns `true` upon success or `JError` object upon failure

create

This method creates a folder and any necessary parent folders.

boolean **create**([**$path** = ''], [**$mode** = 0755])

- *string* **$path**: A path to create from the base path
- *integer* **$mode**: Directory permissions to set for folders created
- *boolean*: Returns `true` upon success

delete

This method deletes a folder.

boolean **delete**(**$path**)

- *string* **$path**: The path to the folder to delete
- *boolean*: Returns `true` upon success

exists

This method determines if a file exists.

boolean **exists**(**$path**)

- *string* **$path**: The path to the folder
- *boolean*: Returns `true` if `$path` is a folder

files

This method returns a list of files located within a folder. The method can list files in all sub-folders if `$recurse` is true or to a maximum depth if `$recurse` is an integer value.

array **files**(**$path**, [**$filter** = '.'], [**$recurse** = false], [**$fullpath** = false] , [**$exclude** = array('.svn', 'CVS')])

- *string* **$path**: The path of the folder to read
- *string* **$filter**: A filter for file names
- *mixed* **$recurse**: If `true` search all sub-folders; integer maximum depth
- *boolean* **$fullpath**: If `true` return the full path to the file
- *array* **$exclude**: Array of file names to exclude from the result
- *array*: Returns an array of files in the specified folder

folders

This method returns a list of folders located within a folder. The method can list all sub-folders if `$recurse` is true or to a maximum depth if `$recurse` is an integer value.

array **folders**(**$path**, [**$filter** = '.'], [**$recurse** = false], [**$fullpath** = false] , [**$exclude** = array('.svn', 'CVS')])

- *string* **$path**: The path of the folder to read
- *string* **$filter**: A filter for folder names
- *mixed* **$recurse**: If `true` search sub-folders; integer maximum depth
- *boolean* **$fullpath**: If `true` return the full path to the folders
- *array* **$exclude**: Array of folder names to exclude from the result
- *array*: Returns an array of folders in the specified folder

listFolderTree

This method returns a list of folders in an associative array suitable for a tree display.

array **listFolderTree**(**$path**, **$filter**, [**$maxLevel** = 3], [**$level** = 0], [**$parent** = 0])

- *string* **$path**: The path of the folder to read
- *string* **$filter**: A filter for folder names
- *integer* **$maxLevel**: Maximum levels to recursively read; defaults to three
- *integer* **$level**: The current level; optional
- *integer* **$parent**: Unique identifier of the parent folder, if any
- *array*: Returns an array of folders in the specified folder

makeSafe

This method sanitizes the path name and makes it safe to use. The method removes any characters that are unsafe to use from a path name.

string **makeSafe**(**$path**)

- *string* **$path**: The full path
- *string*: Returns the sanitized path name

move

This method moves a folder to a new location.

mixed **move**(**$src**, **$dest**, [**$path** = null])

- *string* **$src**: The path to the source folder
- *string* **$dest**: The path to the destination folder
- *string* **$path**: An optional base path to prefix to the file names
- *mixed*: Returns `true` upon success; `JError` message on failure

JFTP

extends `JObject`*, located in* `/joomla/client/ftp.php`

This is the Joomla! FTP client class that provides methods for handling file transfers. For more information about the `JFTP` class refer to Chapter 10, *APIs and Web Services*.

Properties

resource `$_con` = null	Server connection resource
resource `$_datacon` = null	Data port connection resource
string `$_pasv` = null	Passive connection information
string `$_response` = null	Response message
integer `$_timeout` = 15	Timeout limit
integer `$_type` = null	Transfer type
string `$_OS` = null	Native OS type
array `$_autoAscii`	Array to hold ascii format file extensions: `asp`, `bat`, `c`, `cpp`, `csv`, `h`, `htm`, `html`, `shtml`, `ini`, `inc`, `log`, `php`, `php3`, `pl`, `perl`, `sh`, `sql`, `txt`, `xhtml`, `xml`
array `$_lineEndings`	Array to hold native line ending characters: `Unix => "\n",MAC => "\r",WIN => "\r\n"`

Inherited properties

Inherited from `JObject`:

- `JObject::_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor that creates a new instance of `JFTP` object.

Redefines `JObject::__construct();` overridden in descendant classes.

JFTP **__construct**([**$options** = array()])

- *array* **$options**: Optional; Associative array of options to set
- *object*: Returns a new `JFTP` object

Destructor __destruct

Class destructor that closes an existing connection if one exists.

void **__destruct**()

- *void*: No return

chdir

This method changes the current working directory on the FTP server.

boolean **chdir**(**$path**)

- *string* **$path**: Path to change into on the server
- *boolean*: Returns `true` upon success

chmod

This method changes the mode for a path on the FTP server.

boolean **chmod**(**$path**, **$mode**)

- *string* **$path**: Path to change the mode on
- *mixed* **$mode**: Octal value to change mode to, for example - 0777
- *boolean*: Returns `true` upon success

connect

This method establishes a connection to an FTP server.

boolean **connect**([**$host** = '127.0.0.1'], [**$port** = 21])

- *string* **$host**: Optional; Host to connect to, default `'127.0.0.1'`
- *integer* **$port**: Optional; Port to connect on, default `21`
- *boolean*: Returns `true` upon success

create

This method creates an empty file on the FTP server.

boolean **create**(**$path**)

- *string* **$path**: Path to local file to store on the FTP server
- *boolean*: Returns `true` upon success

delete

This method deletes a path (file or folder) on the FTP server.

boolean **delete**(**$path**)

- *string* **$path**: Path to delete
- *boolean*: Returns `true` upon success

get

This method retrieves a file from the FTP server and saves it to a local file.

Redefines `JObject::get()`—returns a property of the object or the default value if the property is not set.

boolean **get**(**$local**, **$remote**)

- *string* **$local**: Path to local file to save remote file as
- *string* **$remote**: Path to remote file to get on the FTP server
- *boolean*: Returns `true` upon success

getInstance

This method returns a reference to the global FTP connector object; if one doesn't exist it will be created. The `$options` associative array can contain two keys:

- `type    => FTP_AUTOASCII | FTP_ASCII | FTP_BINARY,`
- `timeout =>` (integer value for the timeout)

JFTP **getInstance**([**$host** = '127.0.0.1'], [**$port** = 21], [**$options** = null], [**$user** = null], [**$pass** = null])

- *string* **$host**: Optional; host to connect to, default `'127.0.0.1'`
- *integer* **$port**: Optional; port to connect on, default `21`
- *array* **$options**: Optional; associative array of options
- *string* **$user**: Username to use for a connection
- *string* **$pass**: Password to use for a connection
- *object*: Returns a reference to a JFTP object

isConnected

This method determines if the object is connected to an FTP server.

boolean **isConnected**()

- *boolean*: Returns `true` if connected

listDetails

This method lists the contents of a directory on the FTP server.

mixed **listDetails**([**$path** = null], [**$type** = 'all'])

- *string* **$path**: Optional; path to local file to store on the FTP server
- *string* **$type**: Optional; return type [`raw | all | folders | files`]
- *mixed*: If `$type` is raw returns a directory listing; else an array of file names

listNames

This method lists the filenames of the contents of a directory on the FTP server.

string **listNames**([**$path** = null])

- *string* **$path**: Optional; path to local file to store on the FTP server
- *string*: Returns a directory listing

login

This method logs into a server once a connection has been established.

boolean **login**([**$user** = 'anonymous'], [**$pass** = 'jftp@joomla.org'])

- *string* **$user**: Optional; username to login to the server
- *string* **$pass**: Optional; password to login to the server
- *boolean*: If `true` upon success

read

This method reads a file from the FTP server's contents into a buffer.

boolean **read**(**$remote**, **&$buffer**)

- *string* **$path**: Path to remote file to read on the FTP server
- *string* **$buffer**: Buffer variable to read file contents into
- *boolean*: If `true` upon success

reinit

This method reinitializes the server.

boolean **reinit**()

- *boolean*: Returns `true` upon success

rename

This method renames a file or folder on the FTP server.

boolean **rename**(**$from**, **$to**)

- *string* **$from**: Path to change the file or folder from
- *string* **$to**: Path to change the file or folder to
- *boolean*: If `true` upon success

restart

This method restarts a data transfer session at a given byte.

boolean **restart**(**$point**)

- *integer* **$point**: The byte to restart the transfer at
- *boolean*: If `true` upon success

setOptions

This method sets client options.

boolean **setOptions**(**$options**)

- *array* **$options**: An associative array of options to set
- *boolean*: If `true` upon success

store

This method stores a file to the FTP server.

boolean **store**(**$local**, [**$remote** = null])

- *string* **$local**: Path to the local file to store on the FTP server
- *string* **$remote**: Optional; the FTP path where to create the file
- *boolean*: If `true` upon success

syst

This method retrieves the system identifier string from the FTP server. This can be `'MAC'`, `'WIN'`, or `'UNIX'`.

mixed **syst**()

- *mixed*: Returns the system identifier string or `false` if error

write

This method writes a string to the FTP server.

boolean **rename**(**$remote**, **$buffer**)

- *string* **$remote**: Path to write file to
- *string* **$buffer**: Contents to write to the FTP server
- *boolean*: If `true` upon success

JHelp

located in `/joomla/language/help.php`

This is the Joomla! base help system class.

Methods

createSiteList

This method builds a list of the help sites that can be used in a selection option. It parses a XML file to produce the list of help sites and their URLs. The associative array contains three keys: `text`, `value`, and `selected`. If there is a language tag present that will be the value of the `selected` key. The default list of help sites is generated from the file: `[path-to-joomla]/administrator/help/helpsites-15.xml`.

array **createSiteList**(**$pathToXml**, [**$selected** = null])

- *string* **$pathToXml**: The path to an XML file
- *string* **$selected**: A language tag to select (if present)
- *array*: Returns an array of associative arrays of help sites

createURL

This method creates a URL for a specified help file reference.

string **createURL**(**$ref**, [**$useComponent** = false])

- *string* **$ref**: The name of the file (excluding the file extension)
- *boolean* **$useComponent**: If `true` use the help file in component directory
- *string*: Returns a URL string

mkdir

This method creates a directory on the FTP server.

boolean **mkdir**(**$path**)

- *string* **$path**: Directory to create
- *boolean*: Returns `true` upon success

pwd

This method retrieves the current working directory on the FTP server.

string **pwd**()

- *string*: Returns current working directory

quit

This method closes the current connection and quits the application.

boolean **quit**()

- *boolean*: Returns `true` upon success

JLanguage

extends `JObject`, *located in* `/joomla/language/language.php`

This class handles languages and translations.

Properties

boolean `$_debug`	Debug mode
string `$_default`	The default language (en-GB)
string `$_lang`	The language name
array `$_metadata`	Language metadata
array `$_orphans`	Failed translation strings (only maintained during debug)
array `$_paths`	Array of loaded language file paths
array `$_strings`	Associative array of translations
array `$_used`	Array of used strings (only maintained during debug)

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JLanguage` object. Loads the specified language; if `$lang` is not specified the default language, `'en-GB'`, will be loaded.

JApplication **__construct**([**$lang** = null])

- *string* **$lang**: The language to load

_(translate)

This method attempts to translate a string. `$jsSafe`, if `true`, will add slashes to the translated string. If a translation cannot be found the original string will be returned. If debug is enabled, translated strings will be encapsulated by bullet characters, strings translated from a constant will be encapsulated in double exclamation marks, and strings that are not translated will be encapsulated in double question marks.

string _(**$string**, [**$jsSafe** = false])

- *string* **$string**: The string to translate
- *boolean* **$jsSafe**: If `true` make the result JavaScript safe
- *string*: Returns the translated string

exists

This method checks if a language exists in the default language folder. `$basePath` is one level above where the languages reside. This is a simple and quick check for the directory that should contain language files for the given user.

boolean **exists**(**$lang**, [**$basePath** = JPATH_BASE])

- *string* **$lang**: Language to check
- *string* **$basePath**: Optional path to check
- *boolean*: Returns `true` if the language exists

get

This method gets a metadata language property. Common properties include the `name` and `tag`. Redefines `JObject::get();` returns a property of the object or the default value if the property is not set.

mixed **get**(**$property**, [**$default** = null])

- *string* **$property**: The name of the property
- *mixed* **$default**: The default value
- *mixed*: Returns the value of the property

getBackwardLang

This method gets the backward-compatible language name. Used for legacy support.

string **getBackwardLang**()

- *string*: Returns the backward compatible name

getDebug

This method checks if the language object is in debug mode.

boolean **getDebug**()

- *boolean*: Returns `true` if in debug mode

getDefault

This method gets the default language.

string **getDefault**()

- *string*: Returns the default language code

getInstance

This method returns a reference to the global `JLanguage` object. If an instance for the specified `$lang` does not exist it will be created. The method must be invoked as:

```
$browser =& JLanguage::getInstance($lang);
```

JLanguage **&getInstance**(**$lang**)

- *string* **$lang**: Language to load
- *object*: Returns a reference to the global `JLanguage` object

getKnownLanguages

This method returns a two-dimensional array of all the known languages. The array contains keys named the same as the languages with values that are associative arrays of the corresponding language metadata.

array **getKnownLanguages**([**$basePath** = JPATH_BASE])

- *string* **$basePath**: Optional base path to use
- *array*: Returns two-dimensional associative array of languages and metadata

getLanguagePath

This method returns the path to a language. If `$language` is not specified the path will point to all languages.

string **getLanguagePath**([**$basePath** = JPATH_BASE], [**$language** = null])

- *string* **$basePath**: The base path to use
- *string* **$language**: The language name
- *string*: Returns the path to a language or all languages

getLocale

This method returns the language locale. For example, `en-GB, english`.

string **getLocale**()

- *string*: Returns the locale property

getMetadata

This method returns an associative array containing the metadata about the specified language. If `$lang` exists the language metadata is returned as an associative array; if the language does not exist `null` is returned.

mixed **getMetadata**([**$lang**)

- *string* **$lang**: The name of the language
- *mixed*: Returns language metadata or null

getName

This method returns the name of the language.

string **getName**()

- *string*: Returns the official name of the language

getOrphans

This method returns the list of orphaned strings, an array of strings that could not be translated. This information is only collected if the language object is in debug mode.

array **getOrphans**()

- *array*: Returns an array of orphaned strings

getPaths

This method returns an associative array of loaded language file paths. If `$extension` is defined only information about language files that are specific to that extension will be returned.

array **getPaths**([**$extension** = null])

- *string* **$extension**: Optional extension
- *array*: Returns an array of orphan (not translated) strings

getPdfFontName

This method returns the PDF font name.

string **getPdfFontName**()

- *string*: Returns the name of the PDF font to be used

getTag

This method returns the language tag, for example `en-GB`.

string **getTag**()

- *string*: Returns the language tag

getUsed

This method returns an array of strings that were successfully translated. Used strings are strings that were requested to be translated and were either successfully translated or determined to be a constant.

array **getUsed**()

- *array*: Returns the array of used strings

getWinCP

This method returns the Windows locale code page name.

string **getWinCP**()

- *string*: Returns the Windows locale code page name

hasKey

This method determines if a translation exists.

boolean **hasKey**(**$key**)

- *string* **$key**: The key to check
- *boolean*: Returns `true` if the key exists

isRTL

This method determines if the language is written right-to-left (RTL).

boolean **isRTL**()

- *boolean*: Returns `true` if it is a RTL language

load

This method loads a language file. `$extension` is used to identify the extension for which we are loading the language file; this determines where the file is located. `$basePath` is one level above where the languages reside. If a language fails to load, normally because the file does not exist, or is inaccessible, the equivalent default language will be loaded. Note that the new translations are merged with previously loaded translations.

boolean **load**([**$extension** ='joomla'], [**$basePath** = JPATH_BASE], [**$lang** = null], [**$reload** = false])

- *string* **$extension**: The extension for which a language file should be loaded
- *string* **$basePath**: The base path to use
- *string* **$lang**: The language to load, default null for the current language
- *boolean* **$reload**: If true force language to be reloaded
- *string*: Returns path to a language or all languages

setDebug

This method toggles the debug property on or off.

boolean **setDebug($debug)**

- *boolean* **$debug**: Turn debug on or off
- *boolean*: Returns the previous debug value

setDefault

This method sets the default language.

string **setDefault($lang)**

- *string* **$lang**: The default language
- *string*: Returns the previous default language

setLanguage

This method sets the language and loads the language metadata. This does not load the translations; use the `JLanguage::load()` method to load the translations.

string **setLanguage($lang)**

- *string* **$lang**: The language identifier
- *string*: Returns the previous language

transliterate

This method processes a string and replaces all accented UTF-8 characters with unaccented ASCII-7 equivalents.

string **transliterate**(**$string**)

- *string* **$string**: The string to transliterate
- *string*: Returns the transliteration of the string

_load

This *private* method loads a language file and adds its contents to the existing language array. If `$overwrite` is `true` (default) any keys in the new language file that match keys in the existing language array will be overwritten; if `false` the existing keys will not be overwritten.

boolean **_load**(**$filename**, [**$extension** = 'unknown'], [**$overwrite** = true])

- *string* **$filename**: The language path and filename to load
- *string* **$extension**: The name of the extension
- *string* **$overwrite**: If true overwrite matching keys
- *boolean*: Returns `true` on success

_getCallerInfo

This *private* method determines where a method call originated by getting backtrace information.

array **getCallerInfo**()

- *array*: Returns an array of backtrace information

_parseLanguageFiles

This *private* method returns a two-dimensional associative array of all the languages in the path specified by `dir`. The returned associative array contains keys named the same as the languages that contain associative arrays of the corresponding metadata.

array **_parseLanguageFiles**([**$dir** = null])

- *string* **$dir**: The directory of language files
- *array*: Returns two-dimensional associative array of languages & metadata

_parseXMLLanguageFile

This *private* method parses an individual XML language information file and returns an array of metadata.

array **_parseXMLLanguageFile**(**$path**)

- *string* **$path**: The path (including file name) to the XML language file
- *array*: Returns an associative array of metadata

_parseXMLLanguageFiles

This *private* method returns a two-dimensional associative array of all the XML language information files in `$dir`. The array contains keys named the same as the languages, which contain arrays of metadata (for example `filename => metadata array`). Normally there will only be one XML language information file per language.

array **_parseXMLLanguageFiles**([**$dir** = null])

- *string* **$dir**: The directory of language files
- *array*: Returns two-dimensional associative array of languages and metadata

JLanguageHelper

static, located in `/joomla/language/helper.php`

This class performs language functions that are not specific to an individual language.

Methods

createLanguageList

This *static* method builds an array of language options. Each element is an associative array with three keys: `name`, `value`, and `selected`. This array can be used to build a selection list of languages.

array **createLanguageList**(**$actualLanguage**, [**$basePath** = JPATH_BASE], [**$caching** = false])

- *string* **$actualLanguage**: The current language
- *string* **$basePath**: The base path to use
- *boolean* **$caching**: If true use cached response
- *array*: Returns a two-dimensional array of language options

detectLanguage

This method attempts to detect the primary language being used from the HTTP headers. If unable to determine the language from the headers it defaults to `en-GB`.

string **detectLanguage**()

- *string*: Returns the primary language in use

JLDAP

extends `JObject`, *located in* `/joomla/client/ldap.php`

This class provides methods for connecting to an LDAP (Lightweight Directory Application Protocol) server and browsing the contents. For further information on using the `JLDAP` class refer to Chapter 10, *APIs and Web Services*.

Properties

boolean `$auth_method` = null	The authorization method to use
string `$base_dn` = null	The base `DN` (for example, `o=MyDir`)
string `$host` = null	The hostname of the LDAP server
boolean `$negotiate_tls` = null	Negotiate TLS (encrypted communications)
boolean `$no_referrals` = null	No referrals (server transfers)
string `$password` = null	The password to connect to the server
integer `$port` = null	The port of the LDAP server
string `$search_string` = null	The search string
string `$username` = null	The username to connect to the server
string `$sers_dn` = null	The user `DN` (for example: `cn=Users, o=MyDir`)
boolean `$use_ldapV3` = null	Use LDAP version 3
string `$_dn` = null	The current `DN`
mixed `$_resource` = null	The LDAP resource identifier

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JLDAP` object. Redefinition of `JObject::__construct()`; overridden in descendant classes.

JLDAP **__construct**([**$configObj** = null])

- *object* **$configObj**: An object containing configuration variables

add

This method adds an attribute to the specified `DN`. Note that the `DN` must already exist.

boolean **add**(**$dn**, **$entry**)

- *string* **$dn**: The `DN` of the entry to add the attribute
- *array* **$entry**: An array of arrays containing attributes to add
- *boolean*: Returns `true` upon success

anonymous_bind

This method anonymously binds to the LDAP directory.

boolean **anonymous_bind**()

- *boolean*: Returns `true` upon success

bind

This method binds to the LDAP directory. If `$nosub` is `true` the default username will be used, if `false` (default) and `$username` is not null, then use `$username`.

boolean **bind**([**$username** = null], [**$password** = null], [**$nosub** = false])

- *string* **$username**: Optional; the username
- *string* **$password**: Optional; the password
- *integer* **$nosub**: Optional, if `true` use default username
- *boolean*: Returns `true` upon success

close

This method closes the connection to the LDAP server.

void **close**()

- *void*: No return

compare

This method compares an entry and returns `true` if the values are equal, `false` if the values are not equal, or `-1` if an error occurs.

mixed **compare**(**$dn**, **$attribute**, **$value**)

- *string* **$dn**: The `DN` that contains the attribute to compare
- *string* **$attribute**: The attribute whose value is to be compared
- *string* **$value**: The value to compare against the LDAP attribute value
- *mixed*: Returns `true` if equal, `false` if not equal, `-1` on error

connect

This method establishes a connection to the server.

boolean **connect**()

- *boolean*: Returns `true` upon successful connection.

create

This method creates a new `DN`.

boolean **create($dn, $entries)**

- *string* **$dn**: The `DN` where you wish to place the object
- *string* **$entries**: An array of arrays describing the object to add
- *boolean*: Returns `true` upon success

delete

This method deletes the specified `DN` object from the tree.

boolean **delete($dn)**

- *string* **$dn**: The `DN` of the object to be deleted
- *boolean*: Returns `true` upon success

generatePassword

This method generates an encrypted LDAP compatible password.

string **generatePassword($password**, [**$type** = 'md5'])

- *string* **$password**: The clear text password to encrypt
- *string* **$type**: The type of password hash; can be either `'md5'` or `'SHA'`
- *string*: Returns the encrypted password

getDN

This method returns the current `DN`.

string **getDN**()

- *string*: Returns the current `DN`

getErrorMsg

This method returns the error message.

string **getErrorMsg**()

- *string*: Returns the error message

ipToNetAddress

This method converts a dot notation IP address to a net address (for example, an address for Netware, and so on).

string **ipToNetAddress**(**$ip**)

- *string* **$ip**: The IP address (for example - xxx.xxx.xxx.xxx)
- *string*: Returns the net address

LDAPNetAddr

This method extracts a readable network address from the LDAP encoded `$networkaddress` attribute. The method returns an array containing the keys `protocol` and `address`. The `protocol` (address types) supported include IPX, IP, SDLC, Token Ring, OSI, AppleTalk, NetBEUI, Socket, UDP, TCP, UDP6, TCP6, Reserved (12), URL, and Count.

array **LDAPNetAddr**(**$networkaddress**)

- *string* **$networkaddress**: The network address to be converted
- *array*: Returns an array containing the readable address

modify

This method modifies an attribute value.

boolean **modify**(**$dn**, **$attribute**)

- *string* **$dn**: The DN that contains the attribute to modify
- *string* **$attribute**: The attribute whose value is to be modified
- *boolean*: Returns `true` upon success

read

This method reads all or specified attributes of a specified `DN`.

mixed **read**(**$dn**, [**$attribute** = array()])

- *string* **$dn**: The `DN` of the object to be read
- *array* **$attribute**: The attribute values to be read
- *mixed*: Returns array of attribute values or `-1` on error

remove

This method removes the attribute from the specified `DN`.

boolean **remove**(**$dn**, **$attribute**)

- *string* **$dn**: The `DN` that contains the attribute to be removed
- *string* **$attribute**: The attribute to be removed
- *boolean*: Returns `true` upon success

rename

This method renames the `DN` entry.

boolean **rename**(**$dn**, **$newdn**, **$newparent**, **$deleteolddn**)

- *string* **$dn**: The `DN` to be renamed
- *string* **$newdn**: The new `DN` (`cn=newdn`)
- *string* **$newparent**: The full `DN` of the parent
- *boolean* **$deleteolddn**: Delete the old values; default is `true`
- *boolean*: Returns `true` upon success

replace

This method replaces a `DN` entry.

boolean **replace**(**$dn**, **$attribute**)

- *string* **$dn**: The `DN` that contains the attribute to be replaced
- *string* **$attribute**: The attribute to be replaced
- *boolean*: Returns `true` upon success

search

This method performs a LDAP filtered search.

array **search**(**$filters**, [**$dnoverride** = null])

- *array* **$filters**: The search filters (array of strings)
- *string* **$dnoverride**: Optional; the `DN` to override the current `DN`
- *array*: Returns an array of arrays containing the search results

setDN

This method sets the `DN` with some template replacements.

void **setDN**(**$username**, [**$nosub** = false])

- *string* **$username**: The username
- *boolean* **$nosub**: Optional, if `true` use default username
- *void*: No return

simple_search

This method performs a LDAP search using comma separated search strings.

array **simple_search**(**$search**)

- *string* **$search**: Comma separated search strings
- *array*: Returns an associative array of result values

JLog

extends `JObject`*, located in* `/joomla/error/log.php`

This class is designed to build log files based on the W3C specification, available at `http://www.w3.org/TR/WD-logfile.html`. For further information on using the `JLog` class refer to Chapter 12, *Utilities and Useful Classes*.

Properties

resource `$_file`	Log file pointer
string `$_format`	The log format
string `$_path`	The log file path

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JLog` object. Sets the default values for `$path` and `$options`.

Redefinition of `JObject::__construct()`; overridden in descendant classes.

JLog **__construct**(**$path**, **$options**)

- *string* **$path**: The log file path
- *array* **$options**: Log file options

addEntry

This method writes a new entry into a log file. The `$entry` array is an associative array whose keys differ depending on the log file being written.

boolean **addEntry**(**$entry**)

- *array* **$entry**: Log entry array
- *boolean*: Returns `true` if log entry is successfully written

getInstance

This method returns a reference to the global `JLog` object. If it does not exist it will be created. The `$options` parameter is an associative array; currently `JLog` only supports one option: `format`. The `format` option is used to determine the format of log entries. By default this is:

```
"{DATE}\t{TIME}\t{LEVEL}\t{C-IP}\t{STATUS}\t{COMMENT}"
```

If `$path` is not specified the default configuration log path will be used.
This method must be invoked as:

```
$log =& JLog::getInstance();
```

JLog **&getInstance**([**$file** = 'error.php'], [**$options** = null], [**$path** = null])

- *string* **$file**: The log file name; the default is '`error.php`'
- *array* **$options**: Optional array of options
- *string* **$path**: Optional log file path
- *object*: Returns a reference to the global `JLog` object

setOptions

This method sets the format option. The `$options` array is an associative array with one key `format`. The default format is set when a new `JLog` object is created (see the constructor method for the default format string). This method can set a custom format for the log file entry.

boolean **setOptions**(**$options**)

- *array* **$options**: An associative array of options to set
- *boolean*: Returns `true` upon success

_closeLog

This *private* method closes the log file.

boolean **_closeLog**()

- *boolean*: Returns `true` upon success

JMail

extends `PHPMailer`, *located in* `/joomla/mail/mail.php`

This class provides a common interface for sending email from the Joomla! framework. For further information on using the `JMail` class refer to Chapter 10, *APIs and Web Services*.

Inherited properties

Inherited from `PHPMailer`:

- `PHPMailer::$AltBody`
- `PHPMailer::$attachment`
- `PHPMailer::$bcc`
- `PHPMailer::$Body`
- `PHPMailer::$boundary`
- `PHPMailer::$cc`
- `PHPMailer::$CharSet`
- `PHPMailer::$ConfirmReadingTo`
- `PHPMailer::$ContentType`
- `PHPMailer::$CustomHeader`
- `PHPMailer::$Encoding`
- `PHPMailer::$ErrorInfo`
- `PHPMailer::$error_count`
- `PHPMailer::$From`
- `PHPMailer::$FromName`
- `PHPMailer::$Helo`
- `PHPMailer::$Host`
- `PHPMailer::$Hostname`
- `PHPMailer::$language`
- `PHPMailer::$LE`
- `PHPMailer::$Mailer`
- `PHPMailer::$MessageID`
- `PHPMailer::$message_type`
- `PHPMailer::$Password`
- `PHPMailer::$PluginDir`
- `PHPMailer::$Port`
- `PHPMailer::$Priority`
- `PHPMailer::$ReplyTo`

- `PHPMailer::$Sender`
- `PHPMailer::$Sendmail`
- `PHPMailer::$sign_cert_file`
- `PHPMailer::$sign_key_file`
- `PHPMailer::$sign_key_pass`
- `PHPMailer::$SingleTo`
- `PHPMailer::$smtp`
- `PHPMailer::$SMTPAuth`
- `PHPMailer::$SMTPDebug`
- `PHPMailer::$SMTPKeepAlive`
- `PHPMailer::$SMTPSecure`
- `PHPMailer::$Subject`
- `PHPMailer::$Timeout`
- `PHPMailer::$to`
- `PHPMailer::$Username`
- `PHPMailer::$Version`
- `PHPMailer::$WordWrap`

Inherited methods

Inherited from `PHPMailer`:

- `PHPMailer::AddAddress()`
- `PHPMailer::AddAttachment()`
- `PHPMailer::AddBCC()`
- `PHPMailer::AddCC()`
- `PHPMailer::AddCustomHeader()`
- `PHPMailer::AddEmbeddedImage()`
- `PHPMailer::AddReplyTo()`
- `PHPMailer::AddStringAttachment()`
- `PHPMailer::ClearAddresses()`
- `PHPMailer::ClearAllRecipients()`
- `PHPMailer::ClearAttachments()`
- `PHPMailer::ClearBCCs()`
- `PHPMailer::ClearCCs()`
- `PHPMailer::ClearCustomHeaders()`
- `PHPMailer::ClearReplyTos()`
- `PHPMailer::getFile()`
- `PHPMailer::HeaderLine()`

- `PHPMailer::IsError()`
- `PHPMailer::IsHTML()`
- `PHPMailer::IsMail()`
- `PHPMailer::IsQmail()`
- `PHPMailer::IsSendmail()`
- `PHPMailer::IsSMTP()`
- `PHPMailer::MsgHTML()`
- `PHPMailer::Send()`
- `PHPMailer::set()`
- `PHPMailer::SetLanguage()`
- `PHPMailer::Sign()`
- `PHPMailer::SmtpClose()`

Methods

Constructor JMail

Class constructor. Creates a new `JMail` object.

Jmail **JMail**()

addAttachment

This method adds file attachments to the email.

void **addAttachment**(**$attachment**)

- *mixed* **$attachment**: The attachment file name or an array of filenames
- *void*: No return

addBCC

This method adds blind carbon copy (BCC) recipient email addresses to the email.

void **addBCC**(**$bcc**)

- *mixed* **$bcc**: A recipient string or an array of recipients
- *void*: No return

addCC

This method adds carbon copy (CC) recipient email addresses to the email.

void **addCC**(**$cc**)

- *mixed* **$cc**: A recipient string or an array of recipients
- *void*: No return

addRecipient

This method adds recipient email addresses to the email.

void **addRecipient**(**$recipient**)

- *mixed* **$recipient**: A recipient string or an array of recipients
- *void*: No return

addReplyTo

This method adds reply to email addresses to the email. The `$replyto` array can be an array with two elements (`'email@address', 'name'`) or an array of arrays in this format.

void **addReplyTo**(**$replyto**)

- *array* **$replyto**: An email address array
- *void*: No return

getInstance

This method returns a reference to the global instance of a `JMail` object. If it does not exist it will be created. `$id` identifies the `JMail` object to return. If you need an instance to use that does not have the global configuration values use an id string that is not `'Joomla'`.

This method must be invoked as:

```
$mail =& JMail::getInstance();
```

JMail **&getInstance**([**$id** = 'Joomla'])

- *string* **$id**: The id string for the `JMail` instance; optional
- *object*: Returns a reference to an instance of the global `JMail` object

send

This method sends the email. Redefinition of `PHPMailer::Send()`. Creates the message and assigns the Mailer. If the message is not sent successfully then it returns `false`. Use the `PHPMailer::ErrorInfo` variable to view a description of the error.

boolean **send**()

- *boolean*: Returns `true` upon success; `false` upon failure

setBody

This method sets the email body.

void **setBody**(**$content**)

- *string* **$content**: The body of the email
- *void*: No return

setSender

This method sets the sender's email address and name. `$from` can be either an array of two elements (`'email@address', 'name'`) or a string consisting of one email address.

void **setSender**(**$from**)

- *mixed* **$from**: The email address string or array of email address and name
- *void*: No return

setSubject

This method sets the email subject line.

void **setSubject**(**$subject**)

- *string* **$subject**: The subject of the email
- *void*: No return

useSendmail

This method selects sendmail for sending the email. If `$sendmail` is specified it instructs the object to use the sendmail path. If `$sendmail` is not specified it instructs the object to use the PHP `Mail()` function.

boolean **useSendmail**([**$sendmail** = null])

- *string* **$sendmail**: The path to sendmail; optional
- *boolean*: Returns `true` if sendmail is enabled; `false` if PHP `Mail()`

useSMTP

This method selects SMTP for sending email. The first four parameters `($auth`, `$host`, `$user`, and `$pass`) must not be null for SMTP to be used; if any of these are null the PHP `Mail()` function will be used.

void **useSMTP**([**$auth** = null], [**$host** = null], [**$user** = null], [**$pass** = null], [**$secure** = null], [**$port** = 25])

- *string* **$auth**: SMTP Authentication; optional
- *string* **$host**: SMTP host; optional
- *string* **$user**: SMTP username; optional
- *string* **$pass**: SMTP password; optional
- *string* **$secure**: SMTP secure `ssl`, `tls`; optional
- *string* **$port**: SMTP port; optional
- *void*: No return

JMailHelper

static, located in `/joomla/mail/helper.php`

This is an email helper class that performs various mail functions that are not specific to an individual `JMail` object. For further information on using the `JMailHelper` class refer to Chapter 10, *APIs and Web Services*.

Methods

cleanAddress

This method verifies that an email address does not have any extra headers injected into it. It also verifies that it does not contain spaces, semicolons, or commas.

- *access*: public

mixed **cleanAddress**(**$address**)

- *string* **$address**: The email address
- *mixed*: Returns the email address or `false` if injected headers are present

cleanBody

This method cleans any injected headers from the email body. It cleans a multi-line string for use in an email body by removing unsafe characters and any email headers from the string.

string **cleanBody**(**$body**)

- *string* **$body**: The email body string
- *string*: Returns the cleaned email body string

cleanLine

This method cleans single-line strings by removing unsafe characters from the string.

string **cleanLine**(**$value**)

- *string* **$value**: The string to be cleaned
- *string*: Returns the cleaned string

cleanSubject

This method cleans an email subject line string by removing unsafe characters and injected headers from the subject string.

string **cleanSubject**(**$subject**)

- *string* **$subject**: The email subject string
- *string*: Returns the cleaned email subject string

cleanText

This method cleans a multi-line string for use in an email by removing unsafe characters and potential injected header strings.

string **cleanText**(**$value**)

- *string* **$value**: The multi-line string to be cleaned
- *string*: Returns the cleaned multi-line string

isEmailAddress

This method verifies that the email address is in a proper email address format.

boolean **isEmailAddress**(**$email**)

- *string* **$email**: The email address string to be verified
- *boolean*: Returns `true` if the string has the correct format; `false` otherwise

JNode

extends `JObject`, *located in* `/joomla/base/tree.php`

This class works in conjunction with the `JTree` class to create and manage hierarchical tree data structures. For further information on using the `JNode` class refer to Chapter 12, *Utilities and Useful Classes*.

Properties

array `$_children`	An array of children
mixed `$_parent`	The parent node

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`

- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JNode` object. Redefinition of `JObject::__construct();` overridden in descendant classes.

JNode **__construct**()

addChild

This method adds a child node to the `$_children` array.

void **addChild**(**&$node**)

- *JNode* **$node**: The `JNode` child object to add to the array
- *void*: No return

getChildren

This method returns the array of child node objects.

array **&getChildren**()

- *array*: Returns an array of child node objects

getParent

This method returns the parent `JNode` object.

JNode **&getParent**()

- *object*: Returns a reference to the parent `JNode` object

hasChildren

This method returns the number of child nodes.

integer **hasChildren**()

- *integer*: Returns the number of child nodes

setParent

This method sets the node as a parent node.

void **setParent**(**&$node**)

- *JNode* **$node**: The parent `JNode` object to set
- *void*: No return

JPath

located in `/joomla/filesystem/path.php`

This base class is integral to the filesystem library and provides methods for handling paths. For further information on using the `JPath` class refer to Chapter 12, *Utilities and Useful Classes*.

Methods

canChmod

This method checks whether a path's permissions can be changed.

boolean **canChmod**(**$path**)

- *string* **$path**: The path to check
- *boolean*: Returns `true` if the path can have its mode changed

check

This method checks for snooping outside of the file system root. If an attempt is made to access a file system path outside of the Joomla! system, an error is raised and the global `jexit()` function is executed (the application is closed).

void **check($path)**

- *string* **$path**: The file system path to check
- *void*: No return

clean

This method removes all double slashes and backslashes and converts all slashes and backslashes to the defined DS directory separator.

string **clean($path**, **[$ds** = DS**]**)

- *string* **$path**: The file system path to clean
- *string* **$ds**: The directory separator; optional
- *string*: Returns the cleaned path

find

This method searches the directory paths for a given file.

mixed **find($paths**, **$file**)

- *mixed* **$paths**: The path or array of paths to search
- *string* **$file**: The name of the file to search for
- *mixed*: Returns the full path including the file name; `false` if file not found

getPermissions

This method returns the permissions of the file or folder within the specified `$path`.

string **getPermissions($path**)

- *string* **$path**: The path of a file or folder
- *string*: Returns the file system permissions

isOwner

This method determines if the PHP script owns the path.

boolean **isOwner**(**$path**)

- *string* **$path**: The path to check ownership
- *boolean*: Returns `true` if the PHP script owns the path passed

setPermissions

This method changes the permissions recursively on files and directories. If `$filemode` or `$foldermode` is null no change will be made. If any one change fails the entire operation will fail.

boolean **setPermissions**(**$path**)

- *string* **$path**: The root path to begin changing permissions
- *string* **$filemode**: The octal value to change the file permissions
- *string* **$foldermode**: The octal value to change the folder permissions
- *boolean*: Returns `true` if successful

JSimpleXML

extends `JObject`, *located in* `/joomla/utilities/simplexml.php`

This class provides a pure PHP4 implementation of the PHP5 interface `SimpleXML`. As with PHP5's `SimpleXML` it is what it says: simple. Nevertheless, it is an easy way to deal with XML data, especially for read only access.

Because it's not possible to use the PHP5 `ArrayIterator` interface with PHP4 there are some differences between this implementation and that of PHP5:

- The access to the root node has to be explicit in `JSimpleXML`, not implicit as with PHP5. Write `$xml->document->node` instead of `$xml->node`.
- You cannot access `CDATA` using array syntax. Use the method `data()` instead.
- You cannot access attributes directly with array syntax; use `attributes()` to read them.
- Comments are ignored.
- Last but not least, this is not as fast as PHP5 `SimpleXML`—it is pure PHP4.

Note: `JSimpleXML` cannot be used to access sophisticated XML doctypes using datatype ANY (for example, XHTML). With a DOM implementation you can handle this. For further information on using the `JSimpleXML` class refer to Chapter 10, *APIs and Web Services*.

Properties

object `$document` = null	Document element
resource `$_parser` = null	XML parser
array `$_stack` = array()	Current object depth
string `$_xml` = null	XML document

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JSimpleXML` object. Redefinition of `JObject::__construct();` overridden in descendant classes.

JSimpleXML **__construct**([**$options** = null])

getParser

This method gets a reference to an XML parser resource.

resource **getParser**()

- *resource*: Returns a reference to a XML parser resource

importDOM

This method takes a node of a DOM document and converts it into a `JSimpleXML` node which can then be used as a native `JSimpleXML` element. If any error occurs it returns `false`.

mixed **importDOM($node, [$classname** = null])

- *string* **$node**: The DOM document
- *string* **$classname**: Optional; currently ignored
- *mixed*: Returns a `JSimpleXMLElement` object or `false` on error

loadFile

This method converts a well-formed XML document in `$path` into a `JSimpleXMLELement` object. If any errors occur during file access or interpretation it returns `false`.

boolean **loadFile($path, [$classname** = null])

- *string* **$path**: The path to the XML file
- *string* **$classname**: Optional; currently ignored
- *boolean*: Returns `true` upon success; `false` if file is empty

loadString

This method takes well-formed XML string data and returns a `JSimpleXMLELement` object with properties containing the data held within the XML document. If any errors occur it returns `false`.

mixed **loadString($string, [$classname** = null**]**)

- *string* **$string**: The well-formed XML string data
- *string* **$classname**: Optional; currently ignored
- *mixed*: Returns a `JSimpleXMLElement` object or `false` if errors

setParser

This method sets the parser resource handle.

void **setParser($parser**)

- *resource* **$parser**: The XML parser resource handle
- *void*: No return

JSimpleXMLElement

extends `JObject`, *located in* `/joomla/utilities/simplexml.php`

This class stores all of the direct children of an element in the `$children` array. They are also stored by type as arrays. So for example, if a tag had two `<font>` tags as children, there would be a class member called `$font` created as an array. `$font[0]` would be the first font tag, and `$font[1]` would be the second.

To loop through all of the direct children of this object, the `$children` member should be used.

To loop through all of the direct children of a specific tag for this object, it is probably easier to use the arrays of the specific tag names, as explained previously.

For further information on using the `JSimpleXMLElement` class refer to Chapter 10, *APIs and Web Services*.

Properties

array `$_attributes` = array()	Attributes of the XML element
array `$_children` = array()	References to the objects of all direct children of the XML object
string `$_data` = '	The element data
integer `$_level` = 0	The level of the element
string `$_name` = ''	The element name

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JSimpleXMLElement` object. Redefinition of `JObject::__construct();` overridden in descendant classes.

JSimpleXMLElement **__construct**(**$name**, [**$attrs** = array()**]**, **$level**)

- *string* **$name**: The element name
- *array* **$attrs**: The array of element attributes
- *integer* **$level**: The element level

addAttribute

This method adds an attribute to an element. If the attribute exists it will be overwritten with `$value`.

void **addAttribute**(**$name, $value**)

- *string* **$name**: The attribute name
- *string* **$value**: The attribute value
- *void*: No return

addChild

This method adds a direct `JSimpleXMLElement` child object to the element.

JSimpleXMLELement **&addChild**(**$name,** [**$attrs** = array()**]**, [**$level** = null**]**)

- *string* **$name**: The element tag name
- *array* **$attrs**: Optional; array of element attributes
- *integer* **$level**: Optional; the level of the child element
- *object*: Returns a reference to a `JSimpleXMLElement` object

attributes

This method returns the value of a specified attribute or if no attribute is specified returns an array of all attributes.

mixed **attributes**([**$attribute** = null**]**)

- *string* **$attribute**: The name of the attribute
- *mixed*: Returns the attribute value or an array of all attributes

children

This method returns an array of all the children of an element.

array **children**()

- *array*: Returns an array of element children

data

This method returns the element data.

string **data**()

- *string*: Returns element data as a string

getElementByPath

This method returns a reference to a `JSimpleXMLElement` object that represents an element in the document by / separated path.

JSimpleXMLELement **&getElementByPath**(**$path**)

- *string* **$path**: The / separated path to the element
- *object*: Returns a reference to a `JSimpleXMLElement` object

level

This method returns the element level.

integer **level**()

- *integer*: Returns element level as an integer value

map

This method traverses the element tree and maps all the element children.

void **map**(**$callback,** [**$args** = array()**]**)

- *string* **$callback**: The callback function name
- *array* **$args**: Optional; array of arguments
- *void*: No return

name

This method returns the element name.

string **name**()

- *string*: Returns the element name as a string

removeAttribute

This method removes an attribute from an element.

void **removeAttribute**(**$name**)

- *string* **$name**: The name of the attribute
- *void*: No return

removeChild

This method removes a child element from an element.

void **removeChild**(**&$child**)

- *object* **$child**: The `JSimpleXMLElement` child object to be removed
- *void*: No return

setData

This method sets the data of an element.

void **setData**(**$data**)

- *string* **$data**: The string data to set
- *void*: No return

toString

This method returns a well-formed XML string based on a `JSimpleXML` element. Redefines `JObject::toString()`.

string **toString**([**$whitespace** = true])

- *boolean* **$whitespace**: If `true` adds a newline and indentation to the tag
- *string*: Returns a well-formed XML string

JText

static, located in `/joomla/methods.php`

This class translates strings to the correct language using the `JLanguage` class.

Methods

printf

This *static* method works like the PHP `printf()` function, except that `$string` is translated. This method accepts a variable number of parameters. The additional arguments will not be translated. The result is outputted, and the method returns the length of the outputted string. If no additional parameters are specified, a null string will be returned. Refer to the PHP manual for more information: `http://php.net/manual/function.printf.php`.

integer **printf($string, [$args]**)

- *string* **$string**: The string to translate
- *mixed* **$args**: Additional arguments to insert into the string
- *integer*: Returns the length of the translated string

sprintf

This *static* method works like the PHP `sprintf()` function, except that `$string` is translated. This method accepts a variable number of parameters. The additional arguments will not be translated. If no additional parameters are specified, a null string will be returned. Refer to the PHP manual for more information: `http://php.net/manual/function.sprintf.php`.

string **sprintf($string, [$args]**)

- *string* **$string**: The string to translate
- *mixed* **$args**: Additional arguments to insert into the string
- *string*: Returns the translated string

_

This *static* method attempts to translate `$string`. If `$jsSafe` is `true` the method will add slashes to the translated string. See `JLanguage::_()` method for more information.

string _(**$string,** [**$jsSafe** = false])

- *string* **$string**: The string to translate
- *boolean* **$jsSafe**: If `true` add slashes
- *string*: Returns the translated string

JTree

extends `JObject`, *located in* `/joomla/base/tree.php`

This class is used in conjunction with the `JNode` class to build trees that model hierarchical data. For further information on using the `JTree` class refer to Chapter 12, *Utilities and Useful Classes*.

Properties

mixed `$_current` = null	The current working node
mixed `$_root` = null	The root node

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`

- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JTree` object. Creates a new root `JNode` object and sets it as the current node. Redefinition of `JObject::__construct();` class constructor.

JTree **__construct**()

addChild

This method adds a child `JNode` object and attaches it to the current working node. If `$setCurrent` is true it sets the child node to be the current working node.

void **addChild**(**&$node**, [*boolean* **$setCurrrent** = false**]**)

- *object* **$node**: A `JNode` path to search
- *boolean* **$setCurrent**: If `true` sets `$node` to the current working node
- *void*: No return

getParent

This method sets the current working node to the parent node of the current working node.

void **getParent**()

- *void*: No return

reset

This method sets the working node to the root node.

void **reset**()

- *void*: No return

G
Request and Session Handling

This appendix details the Joomla! request and session handling classes including caching and routing. The appendix includes the following classes:

- JCache
- JRequest
- JRouting
- JSession
- JURI

JCache

abstract, extends `JObject`*, located in* `/joomla/cache/cache.php`

This is the Joomla! base cache object. Several subclasses exist for caching different items; subclasses are sometimes referred to as `JCache` types. `JCache` uses `JCacheStorage` subclass objects to store and retrieve cache data.

Direct descendents

`JCacheView`	Cache view type object
`JCachePage`	Cache page type object
`JCacheCallback`	Cache callback type object
`JCacheOutput`	Cache output type object

Properties

JCacheStorage `$_handler`	The cache storage handler object
array `$_options`	An array of options

Inherited properties

Inherited from `JObject`:

- `JObject::_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Deprecated methods

Deprecated Method	**Recommended Alternative**
`setCacheValidation()`	No alternative method recommended

Methods

Constructor __construct

Class constructor. Builds a new `JCache` object. The `$options` associative array can contain the keys `language`, `cachebase`, `defaultgroup`, `caching`, and `storage`.

- `language` is used to create separate caches for different languages.
- `cachebase` is used as the path to the base cache folder.
- `defaultgroup` is the group name used when no group is specified in other methods.
- `caching` is a Boolean value; if `true` caching is enabled. Cached data is identified by an ID and a group. The way the cache is stored differs depending on the chosen storage handler.
- `storage` is a string that defines the default storage handler type.

JCache **__construct**(**$options**)

- *array* **$options**: Associative array of options

clean

This method cleans the cache for a group or groups depending on the mode. If the mode is `'group'` then the method cleans the cache within the group. If the mode is `'notgroup'` the method cleans all the cache not in the group.

boolean **clean**(**$group** = null, [**$mode** = 'group'])

- *string* **$group**: The cache data group
- *string* **$mode**: The mode for cleaning the cache [group | notgroup]
- *boolean*: Returns `true` if successful

gc

This method garbage collects expired cached data.

boolean **gc**()

- *boolean*: Returns `true` if successful

get

This *abstract* method returns cached data identified by an ID and a group.
It returns `false` if no cached data is available.

This method redefines and overrides `JObject::get()`, which returns a property of the object or the default value if the property is not set. Descendant classes redefine and override this method:

- `JCacheView::get()`: Gets the cached view data
- `JCachePage::get()`: Gets the cached page data
- `JCacheCallback::get()`: Executes a cacheable callback if not found in cache otherwise it returns the cached output and the result.

mixed **get**(*string* **$id**, [**$group** = null])

- *string* **$id**: The cache data id
- *string* **$group**: The cache data group
- *mixed*: Returns a cached data group or `false` upon failure

getInstance

Returns a reference to a `JCache` subclass object based on `$type`. If the cache object does not exist it creates a new one. The `$options` are passed to the subclass constructor.

JCache **&getInstance**([**$type** = 'output', [**$options** = array()])

- *string* **$type**: The cache object type to instantiate
- *array* **$options**: The `$options` array
- *object*: Returns reference to a new instance of a `JCache` subclass object

getStores

This *private* method returns an array of names of cache storage handlers that will operate correctly in the current environment.

array **getStores**()

- *array*: Returns an array of storage handler names

remove

This *abstract* method removes a cached data entry identified by `$id` and `$group`.

boolean **remove**(**$id**, [**$group** = null])

- *string* **$id**: The cache data id
- *string* **$group**: The cache data group
- *boolean*: Returns `true` if successful

setCaching

This method enables or disables caching.

void **setCaching**(**$enabled**)

- *boolean* **$enabled**: Set to `true` to enable caching, `false` to disable
- *void*: No return

setLifeTime

This method sets the maximum lifetime of cached items in seconds.

void **setLifeTime**(**$lt**)

- *integer* **$lt**: Cache lifetime in seconds
- *void*: No return

store

This method stores data in the cache by `$id` and `$group`.

boolean **store**(**$data**, **$id**, [**$group** = null])

- *mixed* **$data**: The data to store
- *string* **$id**: The cache data id
- *string* **$group**: The cache data group
- *boolean*: Returns `true` if successful

_getStorage

This *private* method returns a reference to a cache storage handler; if the handler does not exist it will be created.

JCacheStorage **&_getStorage**()

- *object*: Returns a reference to a `JCacheStorage` object

JRequest

static, located in `/joomla/environment/request.php`

This class provides the Joomla! framework with a common interface to access request variables; this includes `$_POST`, `$_GET` and `$_REQUEST`. Variables can be passed through an input filter to avoid injection or returned raw. For further information on using the `JRequest` class refer to Chapter 11, *Error Handling and Security*.

Methods

checkToken

This method checks for a form token in the request. It is used in conjunction with `JHTML::_('form.token')`.

boolean **checkToken**([**$method** = 'post'])

- *string* **$method**: The request method in which to look for the token key
- *boolean*: Returns `true` if the token is found and valid; `false` if not

clean

This *static* method cleans the various HTTP header request hashes of any possible script injection; this includes `$_REQUEST`, `$_FILES`, `$ENV`, `$_GET`, `$_POST`, `$COOKIE`, `$_SERVER`, and `$_SESSION` hashes.

void **clean**()

- *void*: No return

get

This *static* method fetches and returns a clean request hash. The default behavior is to fetch variables depending on the current request method: `GET` and `HEAD` will result in returning `$_GET`; `POST` and `PUT` will result in returning `$_POST`. The source hash can be forced by setting the `$hash` parameter to one of the following values:

- 'post' : `$_POST`
- 'get' : `$_GET`
- 'files' : `$_FILES`
- 'cookie' : `$_COOKIE`
- 'end' : `$_ENV`
- 'server' : `$_SERVER` method through current `$_SERVER['REQUEST_METHOD']`
- 'default' : `$_REQUEST`

mixed **get**([**$hash** = 'default'**], $mask**)

- *string* **$hash**: The request hash source
- *integer* **$mask**: A filter mask for the variable
- *mixed*: Returns a clean request hash

getBool

This is a proxy function for the `getVar()` method. This *static* method fetches and returns a `true` or `false` value from a boolean filtered variable.

boolean **getBool**(**$name**, [**$default** = false**]**, [**$hash** = 'default'**]**)

- *string* **$name**: The variable name
- *boolean* **$default**: The default value if the variable does not exist
- *string* **$hash**: The hash that the variable should come from
- *boolean*: Returns the boolean value of the variable (`true/false`)

getCmd

This is a proxy function for the `getVar()` method. This *static* method fetches and returns a filtered variable; the `cmd` filter only allows characters suitable for use as a command (A-Z, a-z, 0-9, underscores (_), fullstops (.), and dashes (-)) to be returned.

string **getCmd(**$name, **[$default** = ''**], [$hash** = 'default'**]**)

- *string* **$name**: The variable name
- *string* **$default**: The default value if the variable does not exist
- *string* **$hash**: The hash that the variable should come from
- *string*: Returns the request variable

getFloat

This is a proxy function for the `getVar()` method. This *static* method fetches and returns a filtered variable; the `float` filter only allows digits and periods to be returned.

float **getFloat(**$name, **[$default** = 0.0**], [$hash** = 'default'**]**)

- *string* **$name**: The variable name
- *string* **$default**: The default value if the variable does not exist
- *string* **$hash**: The hash that the variable should come from
- *float*: Returns the request variable

getInt

This is a proxy function for the `getVar()` method. This *static* method fetches and returns a filtered variable; the `integer` filter only allows digits to be returned.

integer **getInt(**$name, **[$default** = 0**], [$hash** = 'default'**]**)

- *string* **$name**: The variable name
- *integer* **$default**: The default value if the variable does not exist
- *string* **$hash**: The hash that the variable should come from
- *integer*: Returns the request variable

getMethod

This *static* method returns the request method from `$_SERVER['REQUEST_METHOD']`.

string **getMethod**()

- *string*: Returns the request method

getString

This is a proxy function for the `getVar()` method. This *static* method fetches and returns a filtered variable; the `string` filter deletes 'bad' HTML code if not overridden by `$mask`. A `$mask` can be applied to reduce preprocessing overhead; there are three masks: `JREQUEST_NOTRIM`, `JREQUEST_ALLOWHTML`, and `JREQUEST_ALLOWRAW`. The masks represent integer values 1, 2, and 4 respectively and can be combined using bitwise logic.

string **getString($name**, **[$default** = ''**]**, **[$hash** = 'default'**]**, **[$mask** = 0**]**)

- *string* **$name**: The variable name
- *string* **$default**: The default value if the variable does not exist
- *string* **$hash**: The hash that the variable should come from
- *integer* **$mask**: The mask to override
- *string*: Returns the request variable

getURI

This method returns the full request path.

string **getURI**()

- *string*: Returns the full request path

getVar

This *static* method fetches and returns a request variable. The default behavior is to fetch a variable depending on the current request method: `GET` and `HEAD` will result in returning `$_GET`; `POST` and `PUT` will result in returning `$_POST`.
The source hash can be forced by setting the `$hash` parameter to one of the following values:

- 'post' : `$_POST`
- 'get' : `$_GET`
- 'files' : `$_FILES`
- 'cookie' : `$_COOKIE`
- 'end' : `$_ENV`
- 'server' : `$_SERVER` method through current `$_SERVER['REQUEST_METHOD']`
- 'default' : `$_REQUEST`

A `$mask` can be applied to reduce preprocessing overhead; there are three masks: `JREQUEST_NOTRIM`, `JREQUEST_ALLOWHTML`, and `JREQUEST_ALLOWRAW`. The masks represent integer values 1, 2, and 4 respectively and can be combined using bitwise logic. The `$type` defines the return type for the variable; for valid values see `JFilterInput::clean()`.

mixed **getVar**(**$name**, **[$default** = null**]**, **[$hash** = 'default'**]**,
[$type = 'none'**]**, **[$mask** = 0**]**)

- *string* **$name**: The variable name
- *string* **$default**: The default value if the variable does not exist
- *string* **$hash**: The hash that the variable should come from
- *string* **$type**: The return type for the variable
- *integer* **$mask**: The mask to override
- *mixed*: Returns the request variable

getWord

This is a proxy function for the `getVar()` method. This *static* method fetches and returns a filtered variable; the `word` filter only allows the characters A-Z, a-z, and the underscore (_) to be returned.

string **getWord**(**$name**, [**$default** = ''**]**, [**$hash** = 'default'**]**)

- *string* **$name**: The variable name
- *integer* **$default**: The default value if the variable does not exist
- *string* **$hash**: The hash that the variable should come from
- *string*: Returns the request variable

set

This method sets multiple request variables using `$array`, an associative array of key-value pairs. If `$overwrite` is `true` and an existing key is found, the value is overwritten, if `false` it is left unchanged.

void **set**(**$array**, [**$hash** = 'default'**]**, [**$overwrite** = true**]**)

- *array* **$array**: An associative array of key-value pairs
- *string* **$hash**: The hash where the variable should be set
- *boolean* **$overwrite**: If `true` overwrites the value; `false` no change
- *void*: No return

setVar

This method sets the value of a request variable in a specified request hash and returns the previous value.

string **setVar**(**$name**, [**$value** = null**]**, [**$hash** = 'default'**]**, [**$overwrite** = true**]**)

- *string* **$name**: The variable name
- *string* **$value**: The value to set
- *string* **$hash**: The hash where the variable should be set
- *boolean* **$overwrite**: If `false` returns the existing request variable value
- *string*: Returns the previous request variable value

_cleanArray

This *private* method checks an array for banned keys that can be used for PHP injections. If `$globalise` is true the key-value pair will be added to the `GLOBALS` array.
If no banned keys are found the array remains unchanged otherwise the application exits.

void **_cleanArray**(**&$array**, [**$globalise** = false**]**)

- *array* **$array**: The array to clean
- *boolean* **$globalise**: If `true` add the array to the `GLOBALS`
- *void*: No return

_cleanVar

This *private* method checks an array for banned keys that can be used for PHP injections. If `$globalise` is true the key-value pair will be added to the `GLOBALS` array. If no banned keys are found the array remains unchanged otherwise the application exits. The filter bit mask uses bitwise logic to filter the input variable; it can take the following values:

- `JREQUEST_NOTRIM`

  An integer value of 1. If this flag is set and the input is a string, leading and trailing whitespace will be trimmed. If no bits other than the 1 bit are set a strict filter is applied.
- `JREQUEST_RAW`

  An integer value of 2. If this flag is set no filtering is performed and higher bits are ignored.
- `JREQUEST_ALLOWHTML`

  An integer value of 4. If this flag is set HTML is allowed but passed through a safe HTML filter first. If set no more filtering is performed.

void **_cleanVar**(**$var**, **$mask**, [**$type** = null**]**)

- *mixed* **$var**: The input variable
- *integer* **$mask**: The filter bit mask
- *string* **$type**: The variable type
- *void*: No return

_stripSlashesRecursive

This *private* method strips slashes recursively on an array. It returns the array with backslashes stripped off (for example, (\') becomes ('), double backslashes (\\ ) are made into a single backslash (\ ).

array **_stripSlashesRecursive**(**&$array**)

- *array* **$value**: The array to strip slashes
- *array*: Returns the input array with extra backslashes stripped away

JRoute

static, located in `/joomla/methods.php`

This class handles internal URIs. For further information on using the `JRoute` class refer to Chapter 5, *Component Design*.

Methods

_

This method translates an internal URL into a human readable URL. The `$ssl` parameter specifies the secure state of the URI:

- 1 : Make URI secure using global secure site URI
- 0 : Leave URI in the same secure state as it was passed to the function
- -1 : Make the URI unsecure using the global unsecure site URI

string _(**$url**, [**$xhtml** = true], [**$ssl** = null])

- *string* **$url**: An absolute or relative URI to a Joomla resource
- *boolean* **$xhtml**: If `true` replace all `&` with `&amp;` for XML compliance
- *integer* **$ssl**: Secure state for the URI
- *string*: Returns human readable URL

JSession

extends `JObject`, *located in* `/joomla/session/session.php`

This class provides access to session state values as well as session level settings and lifetime management methods. Based on the standard PHP session handling mechanism it provides more advanced features such as expire timeouts. For further information on using the `JSession` class refer to Chapter 4, *Extension Design*.

Properties

integer `$_expire` = 15	The length of time before a session expires (in minutes).
array `$_security` = array()	Security session validation options. Can include the keys `fix_browser` and `fix_adress` (note that `fix_adress` is not a typo).
string `$_state` = 'active'	State of the session (`active`, `expired`, `destroyed`, or `error`).
Object `$_store`	`JSessionStorage` handler.

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`
- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JSession` object. `$store` is the storage handler type, normally `database`. Redefinition of `JObject::__construct();` class constructor, overridden in descendant classes.

JSession **__construct**([**$store**= 'none'**]** , [**$options** = array()**]**)

- *string* **$store**: The session storage handler type
- *array* **$options**: An associative array of storage handler options

Destructor __destruct

Class destructor. Closes the session.

void **__destruct**()

clear

This method removes a value from the session.

mixed **clear(\$name**, **[\$namespace** = 'default'**]**)

- *string* **$name**: Name of the variable
- *string* **$namespace**: Namespace to remove the value from
- *mixed*: Returns the cleared value

close

This method writes session data and closes the session gracefully. Session data is usually stored after a script terminates without the need to call `JSession::close()`,but as session data is locked to prevent concurrent writes only one script may operate on a session at any time. When using framesets together with sessions the frames will load one at a time due to this locking. The time needed to load all the frames can be reduced by ending the session as soon as all changes to session variables are complete.

void **close**()

- *void*: No return

destroy

This *static* method frees all session variables and destroys all data registered to a session. This method resets the `$_SESSION` variable and destroys all of the data associated with the current session in its storage (file or database). It forces a new session to be started after the method is called. It does not remove the session `cookie` or session `id`. This method is equivalent to the PHP `session_destroy()` function.

void **destroy**()

- *void*: No return

fork

This method creates a new session and copies the data from the existing session to the new session.

boolean **fork**()

- *boolean*: Returns `true` upon success

get

This *static* method retrieves a value from the session. If the value is not set `$default` is returned. Redefines `JObject::get();` returns a property of the object or the default value if the property is not set.

mixed **&get($name**, **[$default** = null**]**, **[$namespace** = 'default'**]**)

- *string* **$name**: Name of the variable
- *mixed* **$default**: Default value of a variable if it is not set
- *string* **$namespace**: Namespace to remove the value from
- *mixed*: Returns the value of the named variable

getExpire

This method returns the number of minutes remaining before the session expires.

integer **getExpire**()

- *integer*: Returns the session expiration time in minutes

getId

This method returns the session ID; it returns null if the session has been destroyed.

mixed **getId**()

- *mixed*: Returns the session ID or null

getInstance

This method returns a reference to the global `JSession` object. If the session object does not exist it creates it. The `$handler` contains the type of storage handler; supported types include `APC`, `database`, `eaccelerator`, `memcache`, and `xcache`. The method must be invoked as:

```
$session =& $JSession::getInstance();
```

JSession **&getInstance**(**$handler**, **[$options** = array()**]**)

- *string* **$handler**: The storage handler type
- *string* **$options**: An associative array of storage handler options
- *object*: Returns a reference to a `JSession` object

getName

This method returns the session name; it returns null if the session has been destroyed.

mixed **getName**()

- *mixed*: Returns the session name string or null

getState

This method returns the state of the session (`active`, `expired`, `destroyed`, or `error`).

string **getState**()

- *string*: Returns the state of the session

getStores

This method returns the names of the session storage handlers that work in the current environment.

array **getStores**()

- *string*: Returns an array of session storage handler names

getToken

This method returns a session token. If a token has not been generated one will be generated. Tokens are random alphanumeric strings that can be used to increase the security of requests. They are used to secure forms from spamming attacks. Once a token has been generated the system will check the post request to see if it is present; if it is not it will invalidate the session.

string **getToken**([**$forceNew** = false])

- *boolean* **$forceNew**: If `true` force a new token to be created
- *string*: Returns a session token

has

This method checks if a value exists in the session store.

boolean **has**(**$name**, [**$namespace** = 'default'])

- *string* **$name**: The name of the variable
- *string* **$namespace**: The namespace of the variable
- *boolean*: Returns `true` if the value is set in the session

hasToken

This method compares the session token with `$tCheck`. If the tokens do not match and `$forceExpire` is `true` the session will be expired.

boolean **hasToken**(**$tCheck**, [**$forceExpire** = true])

- *string* **$tCheck**: The hashed token to check the session token against
- *boolean* **$forceExpire**: Force the session to expire if the token is invalid
- *boolean*: Returns `true` if the tokens match

isNew

This method determines if the session was created during this request.

boolean **isNew**()

- *boolean*: Returns `true` upon success

restart

This method restarts an expired or locked session. This will remove any existing session data.

boolean **restart**()

- *boolean*: Returns `true` upon success

set

This method sets a value in a session store.

Redefines `JObject::set()` that modifies a property of the object creating it if it does not already exist.

mixed **set($name**, **$value**, **[$namespace** = 'default'**]**)

- *string* **$name**: The name of the variable to set
- *mixed* **$value**: The value of the variable to set
- *string* **$namespace**: The namespace of the variable
- *mixed*: Returns the previous value of a variable

_createId

This *private* method creates a new session ID.

string **_createId**()

- *string*: Returns a new session ID

_createToken

This *private* method creates a new token.

string **_createToken**()

- *string*: Returns a new token string.

_setCounter

This *private* method increments the session counter. This method must only be invoked once per request.

boolean **_setCounter**()

- *boolean*: Returns `true` upon success

_setOptions

This *private* method sets session options. The `$options` associative array can include the keys `name`, `id`, `expire`, and `security`.

boolean **_setOptions**(**&$options**)

- *array* **$options**: The session options
- *boolean*: Returns `true` upon success

_setTimers

This *private* method sets the session timers including the session start time, the last request time, and the current request time.

boolean **_setTimers**()

- *boolean*: Returns `true` upon success.

_start

This *private* method starts a new session or continues a previous session; this method is equivalent to the PHP `session_start()` function.

boolean **_start**()

- *boolean*: Returns `true` upon success

_validate

This *private* method validates the session. If the session has exceeded the maximum expiry time, the session state will be changed to `expired`. It checks that the client address and browser match the security array, if they are defined in the security array.

boolean **_validate**([**$restart** = false])

- *boolean* **$restart**: If `true` reactivate the session
- *boolean*: Returns `true` upon success

JURI

extends `JObject`, *located in* `/joomla/environment/uri.php`.

This class serves two purposes:

1. To parse a URI and provide a common interface for the Joomla! framework to access and manipulate a URI
2. To obtain the URI from the server without regard to the server

For further information on using the `JURI` class refer to Chapter 2, *Getting Started*.

Properties

string `$_fragment` = null	URI fragment (internal document location)
string `$_host` = null	Host
string `$_pass` = null	URI password (not the Joomla! user's password)
string `$_path` = null	Path
string `$_port` = null	Port number
string `$_query` = null	GET query
string `$_scheme` = null	URI scheme (for example, HTTP)
string `$_uri` = null	URI
string `$_user` = null	URI username (not the Joomla! user's name)
array `$_vars` = array()	Query variable hash

Inherited properties

Inherited from `JObject`:

- `JObject::$_errors`

Inherited methods

Inherited from `JObject`:

- `JObject::JObject()`
- `JObject::__construct()`
- `JObject::get()`
- `JObject::getError()`
- `JObject::getErrors()`
- `JObject::getProperties()`

- `JObject::getPublicProperties()`
- `JObject::set()`
- `JObject::setError()`
- `JObject::setProperties()`
- `JObject::toString()`

Methods

Constructor __construct

Class constructor. Builds a new `JURI` object. If `$uri` is specified it will be parsed and the object properties populated. Redefinition of `JObject::__construct();` class constructor.

JTree **__construct**([**$uri** = null**]**)

base

This *static* method returns the base URI for the request. If `$pathonly` is false the scheme, host, and port properties will be prepended to the base URI.

string **base**([**$pathonly** = false**]**)

- *boolean* **$pathonly**: If `true` return the base path only; default is false
- *string*: Returns the base URI

buildQuery

This method builds a query from an associative array and returns the query string. The method uses `$akey` to recursively build the query from any nested arrays.

string **buildQuery**(**$params**, [**$akey** = null**]**)

- *array* **$params**: The source associative array to build the query from
- *string* **$akey**: A nested associative array key
- *string*: Returns the query string

current

This method returns the URI of the current location including the scheme, host, port and path but minus the query.

string **current**()

- *string*: Returns the current URI

delVar

This method removes a variable from the URI query string.

void **delVar**(**$name**)

- *string* **$name**: The name of the variable to remove
- *void*: No return

getFragment

This method returns the URI anchor string (everything after the hash '#').

string **getFragment**()

- *string*: Returns the URI anchor string

getHost

This method returns the URI host name or IP address. This does not include the path to the resource. If no host name or IP address was specified the method returns null.

mixed **getHost**()

- *mixed*: Returns the URI host name, IP address or null

getInstance

This method returns a reference to a global instance of a JURI object. If an instance does not exist one will be created. If `$uri` is not specified the URI will be constructed based on the current request. This method must be invoked as follows:

```
$uri =& JURI::getInstance($uri);
```

JURI **&getInstance**([**$uri** = 'SERVER'**]**)

- *string* **$uri**: Optional URI to parse; if null uses script URI
- *object*: Returns a reference to a `JURI` object

getPass

This method returns the URI password. This is a part of the scheme authorization; it is not the same as Joomla! authorization. If no password was specified the method returns null.

mixed **getPass**()

- *mixed*: Returns the URI password or null

getPath

This method returns the URI path. This does not include the host name.

string **getPath**()

- *string*: Returns the URI path

getPort

This method returns the URI port. If it is the default port for the scheme or no port was specified this method returns null.

mixed **getPort**()

- *mixed*: Returns the URI port or null

getQuery

This method returns the URI query. If `$toArray` is false it will return a string, if true it will return an associative array.

mixed **getQuery**([**$toArray** = false**]**)

- *boolean* **$toArray**: If `false` returns a string; `true` an associative array
- *mixed*: Returns a string or associative array containing the URI query

getScheme

This method returns the URI scheme (protocol), for example, `http`, `https`, `ftp`, and so on.

string **getScheme**()

- *string*: Returns the URI scheme

getUser

This method returns the URI username. This is part of HTTP authorization; it is not the same as Joomla! authorization or Joomla! user's name. If no username was specified the method returns null.

mixed **getUser**()

- *mixed*: Returns the URI username or null

getVar

This method returns the value of a named URI query variable or `$default` if the variable is not set.

mixed **getVar**([**$name** = null**]**, [**$default** = null**]**)

- *string* **$name**: The query variable name
- *mixed* **$default**: The default value if the variable is not set
- *mixed*: Returns the query value

isInternal

This method determines if the supplied URL is internal.

boolean **isInternal**(**$url**)

- *string* **$url**: The URL to check
- *boolean*: Returns `true` if internal URL

isSSL

This method determines whether the current URI scheme is `https`.

boolean **isSSL**()

- *boolean*: Returns `true` if scheme is `https`

parse

This method parses the `$uri` and populates the class properties.

boolean **parse**(**$uri**)

- *string* **$uri**: The URI string to parse
- *boolean*: Returns `true` upon success

root

This *static* method returns the root URI for the request. If `$pathonly` is `true` only the path is returned; if `false` prepend the scheme, host, and port information to the path.

string **root**([**$pathonly** = false**]**, [**$path** = null**]**)

- *boolean* **$pathonly**: If true only the URI path is returned
- *string* **$path**: The URI path
- *string*: Returns the root URI string

setFragment

This method sets the URI anchor string (everything after the hash '#').

void **setFragment**(**$anchor**)

- *string* **$anchor**: The URI anchor string
- *void*: No return

setHost

This method sets the URI host name. This does not include the path to the resource.

void **setHost**(**$host**)

- *string* **$host**: The URI host name or IP address
- *void*: No return

setPass

This method sets the URI password. This is part of the scheme authorization and is not the same as Joomla! authorization.

void **setPass**(**$pass**)

- *string* **$pass**: The URI password
- *void*: No return

setPath

This method sets the URI path.

void **setPath**(**$path**)

- *string* **$path**: The URI path
- *void*: No return

setPort

This method sets the URI port.

void **setPort(\$port)**

- *string* **$port**: The URI port
- *void*: No return

setQuery

This method sets the URI query. This can be done using a query string or an associative array.

void **setQuery**(**$query**)

- *mixed* **$query**: The URI query string or associative array
- *void*: No return

setScheme

This method sets the URI scheme (protocol) for example - `http`, `https`, `ftp`, and so on.

void **setScheme**(**$scheme**)

- *string* **$scheme**: The URI scheme
- *void*: No return

setUser

This method sets the URI username. This is part of the scheme authorization and is not the same as Joomla! authorization.

void **setUser**(**$user**)

- *string* **$user**: The URI username
- *void*: No return

setVar

This method adds a variable and a value to the URI query string replacing the value if the variable already exists. It returns the previous value.

string **setVar**(**$name**, **$value**)

- *string* **$name**: The name of the query variable to set
- *string* **$value**: The value of the query variable
- *string*: Returns the previous value of the query variable

toString

This method returns the URI in string format including the defined parts. The default value for `$parts` includes all of the possible parts of the URI (`'scheme','user','pass','host','port','path','query','fragment'`). The order of elements in `$parts` is not important. Redefines `JObject::toString()` — object-to-string conversion.

string **toString**(**[$parts** = array()**]**)

- *array* **$parts**: An array specifying the parts to render
- *string*: Returns the rendered URI string

H
XML Manifest File

The XML manifest file details everything the installer needs to know about an extension. Any mistakes in the file may result in partial or total installation failure. XML manifest files should be saved using UTF-8 encoding.

The following table describes the tags you can use in your XML manifest files:

install (Root tag)

There are two different `install` tags. The root tag called install identifies the type of extension and the Joomla! version for which the extension has been written.

Example	`<install type="component" version="1.5">`
	`     <!-- sub-tags -->`
	`</install>`	

Attributes	type	Type of extension, normally component, module or plugin
	version	Version of Joomla! the extension is for
Sub-tags	administration, author, authorEmail, authorUrl, copyright, creationDate, description, files*, install, installfile, languages*, license, media*, name, params, uninstall, uninstallfile, version	
Extensions	components, modules, plugins	

administration

Container for all the component's backend tags. This tag is required even if your component needs no backend tags.

Example	`<administration />`
Sub-tags	files, languages, media, menu, submenu
Extensions	components

author

The author's name.	
Example	`<author>John Smith</author>`
Extensions	components, modules, plugins

authorEmail

The author's email address.	
Example	`<authorEmail>johnsmith@example.org</authorEmail>`
Extensions	components, modules, plugins

authorUrl

The author or component's website address.	
Example	`<authorUrl>http://www.example.org</authorUrl>`
Extensions	components, modules, plugins

copyright

Copyright notice.	
Example	`<copyright>Copyright me!</copyright>`
Extensions	components, modules, plugins

description

Component description.	
Example	`<description>Description of the component</description>`
Extensions	components, modules, plugins

file

SQL file to execute. This is a sub-tag of the `<sql>` tag.		
Example	`<file charset="utf8" driver="mysql">install.sql</file>`	
Attributes	charset	UTF-8
	driver	Database driver name, normally mysql (mysql and mysqli are synonymous in this context).
Extensions	components	

files

Groups files and folders together to make things a bit tidier. To prevent confusion we normally use the optional `folder` attribute to make the archive tidier. This tag has two sub-tags, `filename` and `folder`, which can be used zero to many times.

Example	`<files folder="site"><!-- sub-tags --></files>`	
Attributes	folder	Folder in the archive where the files reside.
Sub-tags	Filename, folder	
Extensions	components, modules, plugins	

filename

Defines a file to be copied into the specified folder. The folder where the file is located can be added if folder has not been specified.

Example	`<filename>example.php</filename>`
	`<filename>models/example.php</filename>`

Extensions	components, modules, plugins

folder

Defines folders we want to copy; if a folder has subfolders and files, we do not have to specify these.

Example	`<folder>afolder</folder>`
Extensions	components, modules, plugins

install

Database installation options. Do not confuse this with the root install tag!

Example	`<install><sql><file></file></sql></install>`
	`<install>`
	`<queries><query></query></queries>`
	`</install>`

Sub-tags	queries, sql
Extensions	components

installfile

File to execute when installing the component. The file can optionally include a function called `com_install()`, returning `true` on success. This is only required if you want to perform additional processing during installation.

Example	`<installfile>install.php</installfile>`
Extensions	components

language

Language tags define a language INI file. The tag includes the attribute `tag`; this is used to identify the language.

Example	`<language tag="en-GB">en-GB.com_example.ini</language>`	
Attributes	tag	Language tag.
Extensions	components, modules, plugins	

languages

Language files. If any of the language files already exist, they will not be overwritten. This tag has one sub-tag, `language`. Each `language` tag defines a language INI file. The `language` tag must include the attribute `tag`; this is used to identify the language.

Example

```
<languages folder="languages">
     <!-- sub-tags -->
</languages>
```

Attributes	folder	Folder in the archive where the files reside.
Sub-tags	language	
Extensions	components, modules, plugins	

license

License agreement.

Example	`<license>GNU GPL</license>`
Extensions	components, modules, plugins

media

Media files to be copied to the root Joomla! images folder.

Example	`<media destination="stories"><!—sub tags --></media>`	
Attributes	destination	Destination folder within the Joomla! images folder.
	folder	Source folder.
Sub-tags	filename	
Extensions	components, modules, plugins	

menu

Backend menu items.		
Example	`<menu>Menu Name</menu>`	
Attributes	[act]	Optional link parameter.
	[controller]	Optional link parameter.
	[img]	Optional location of menu item image.
	[layout]	Optional link parameter.
	[link]	Optional URI link.
	[sub]	Optional link parameter.
	[task]	Optional link parameter.
	[view]	Optional link parameter.
Extensions	components	

name

The extension name (Required). Must be the actual extension name; can contain spaces and capital letters. The name will be converted to lowercase and spaces removed.	
Example	`<name>My Example</name>`
Extensions	components, modules, plugins

param

A parameter. How this tag is used depends upon the type of parameter we are defining; a complete description of these types and their attributes is available in *Appendix B, Parameters (Core JElement)*.	
Example	`<param type="text" name="foobar" label="Foobar"/>`
Extensions	components, modules, plugins

params

Group tag of one or more `param` tags.		
Example	`<params><!—sub tags --></params>`	
Attributes	addPath	Directory where custom `JElement` subclasses can be found.
Sub-tags	param	
Extensions	components, modules, plugins	

queries

Group tag of one or more query tags.	
Example	`<queries><!—sub tags --></queries>`
Sub-tags	query
Extensions	components

query

SQL queries to execute.

Example

```
<query>CREATE TABLE `#__myextension` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(255) NOT NULL default '',
  PRIMARY KEY  (`id`)
) CHARACTER SET `utf8` COLLATE `utf8_general_ci`
</query>
```

Sub-tags	query
Extensions	components

submenu

Backend sub-menu.	
Example	`<submenu><!— sub tags --></submenu>`
Sub-tags	menu
Extensions	components

sql

SQL files to execute.	
Example	`<sql><!— sub tags --></sql>`
Sub-tags	file
Extensions	components

uninstall

Database un-installation options. Do not confuse this with the root install tag!

Example	`<uninstall><!– sub tags --></uninstall>`
Sub-tags	Queries, sql
Extensions	components

uninstallfile

File to execute when uninstalling the component. The file can optionally include a function called `com_uninstall()`, returning `true` on success. This is only required if you want to perform additional processing during un-installation.

Example	`<uninstallfile>uninstall.php</uninstallfile>`
Extensions	components

version

Extension version. Most extensions use three digits in the form `major.minor.patch`; version 1.0.0 normally denotes the first stable release.

Example	`<version>1.0.0</version>`
Extensions	components, modules, plugins