



**AccordionTreeMenu** version 1.0.0

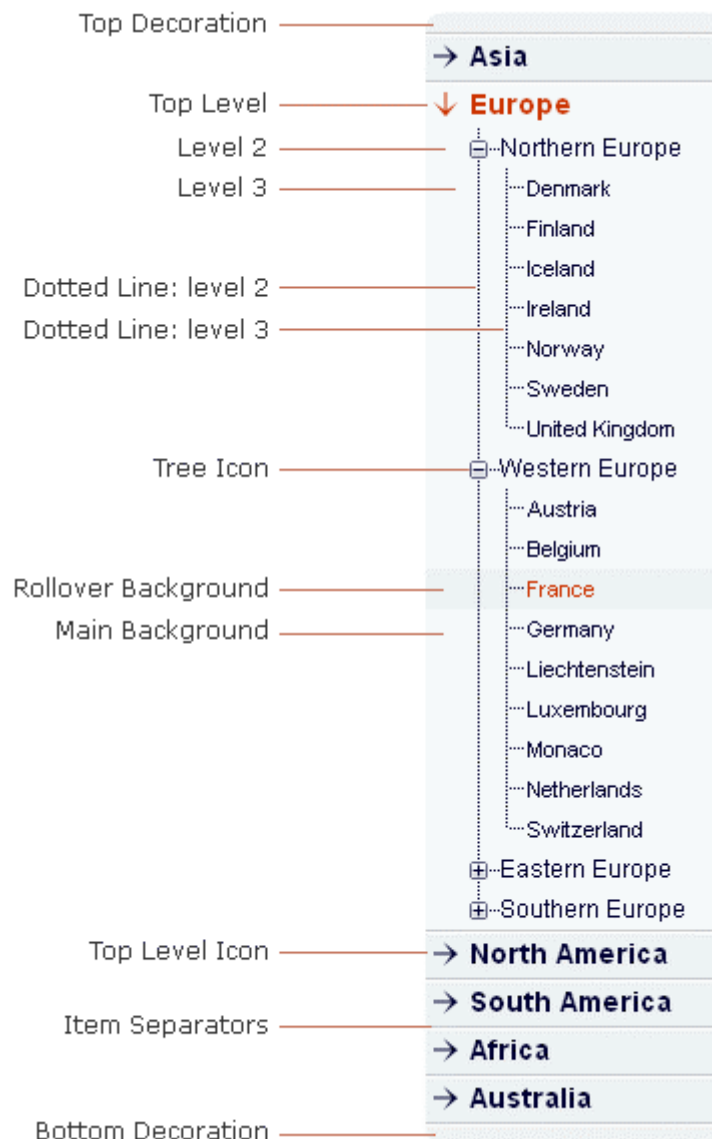
**ADVANCED COMPONENT  
FOR FLASH 8, FLASH CS3/CS4**

## Overview

The Accordion Tree Menu is a Flash component that represents a tree menu structure in an accordion fashion. It allows you to implement a hierarchical menu into your website navigation system.

The component provides with great usability supporting most complete feature set:

- multilevel structure (up to 3 levels) and unlimited menu items
- menu data is loaded using XML
- component can be completely [configured through the XML file](#)
- separate style declaration for each of three levels of menu structure
- alpha transparency support for all backgrounds
- adding URLs or custom Flash functions to any menu click event (through the XML file)
- adding sounds to click or rollover event (through the XML file or authoring parameters in the Component inspector).



The structure of Accordion Tree Menu includes three possible levels:

1. *Top level* — consists of items expanding and folding in accordion fashion. Move your mouse over a top level item, and nested second level sub-menu (if one exists) drops down. Only one top level item can be open at a time.
2. *Second level* (optional) — consists of items expanding and folding in tree fashion. Click on tree icon, and nested third level sub-menu moves up or down. Multiple second level items can stay open at the same time.
3. *Third level* (optional) — the deepest level of Accordion Tree Menu hierarchy.

## Using Accordion Tree Menu

The Accordion Tree Menu can be used to create the professionally looking navigation system for websites of any complexity. Using parameter settings in the authoring environment and ActionScript methods, properties and events available for the component, you can build advanced and full-featured navigation systems. Elegant design and excellent dynamics will create unique and attractive appearance for different types of Flash and HTML-based web applications.

Use Accordion Tree Menu to create any of the following:

- Accordion menus
- Tree structure menus
- Combinations of accordion menu and tree structure menu
- Vertical rollover menus

This is not the complete list of possible usages, just ones that are likely to be used most often. Your creative power and imagination can greatly extend the scope of the Accordion Tree Menu component.

### Accordion Tree Menu parameters

You can set the following authoring parameters for each Accordion Tree Menu component instance in the Component inspector:

**Decoration Height:** Indicates the height of the top and bottom decorations, in pixels. This parameter accepts positive integers (including 0) but not greater than the value of [Item Height \(level 1\)](#) parameter. If Decoration Height is equal to 0, then no decoration will be drawn. The default value is 0.

**Decoration Rounded:** Specifies which corners of the menu decoration to be drawn rounded. This parameter can be one of four predefined values: "left", "right", "all", "none". The default value is "all".



**Dotted Line (level 2):** Indicates whether the dotted lines for second level sub-menus to be drawn ("true") or not ("false"). If Dotted Line (level 2) parameter is set to "false", then no tree icons are drawn, and each second level menu item (if it has nested third level sub-menu) stays open permanently. The default value is "true".

**Dotted Line (level 3):** Indicates whether the dotted lines for third level sub-menus to be drawn ("true") or not ("false"). The default value is "true".

**Easing Class:** Specifies which easing class to be applied when opening or closing a second level menu item. This parameter can be either "regular" or "none". The default value is "regular".

**Easing Method:** Specifies which easing method to be applied when opening or closing a second level menu item. This parameter can be one of three predefined values: "easeIn", "easeOut" or "easeNone"; the default value is "easeOut".

**Initial State Return:** This parameter instructs the menu how to behave when the mouse pointer rolls out of an open top level item outside the menu area. There are two values available:

*false* The open top level item remains open.

*true* The menu returns to its initial state. If Selected State Mode is set to "true" and the selectedIndex property has a definite value (see [Selected State Mode](#) for more information), then the top level item including nested item, which index is equal to the selectedIndex value, will be opened. If there is no item selected, the open top level item will be closed.

The default value is "true".

**Internal Selection Mode:** Turns on/off the component's ability ("true" or "false") to make an item selected when it is clicked. That means the Accordion Tree Menu can change the value of the selectedIndex property inside a movie clip without reloading HTML page. This option is useful, first of all, for Flash-based web applications.

*Note:* For this parameter to be activated, Selected State Mode has to be set to "true".

The default value is "false".

**Item Height (level 1):** Indicates the height of each top level item, in pixels. The value of this parameter is controlled programmatically, considering specified font size and font family. The default value is 24.

**Item Height (level 2):** Indicates the height of each second and third level item, in pixels. The value of this parameter is controlled programmatically, considering specified font size and font family. The default value is 20.

**Item Indent (level 2):** This parameter indicates whether to indent the second level items. The amount of indent is determined programmatically. The default value is "true".

*Note:* If both Item Indent (level 2) and [Top Level Icons](#) parameters are set to "false", then neither tree icons nor dotted lines for second level sub-menus are drawn, and each second level menu item (if it has nested third level sub-menu) stays open permanently.

**Item Indent (level 3):** This parameter indicates whether to indent the third level items. The amount of indent is determined programmatically. The default value is "true".

*Note:* If Item Indent (level 3) is set to "false", then no dotted lines for third level sub-menus are drawn.

**Item Separators (level 1):** Determines whether to draw separator lines between the top level menu items. The default value is "true".

**Item Separators (level 2):** Determines whether to draw separator lines between the second level menu items and between the third level menu items. The default value is "false".

**Left Margin:** Indicates the left margin of the menu, in pixels. This parameter accepts values from 0 to 100. The default value is 0.

**Selected State Mode:** Turns on/off the component's ability ("true" or "false") to make an item selected if the selectedIndex property has a definite value.

*Note:* The [selectedIndex](#) property has a definite (not undefined) value in the following three cases:

1. The variable selIndex is passed to a SWF (containing Accordion Tree Menu component instance) through HTML tags (see [example](#) for more information). If the component instance is not at the top level of a movie, you must specify the selIndex variable at the component's level.

2. The Internal Selection Mode parameter is set to "true" and a menu item has been clicked.
3. The `setItemSelected()` method has been called.

The default value is "true".

**Sound onClick:** An object type parameter that contains two values to be set: source and volume.

*source* This parameter is the path to an external mp3 file loaded into a movie clip and attached to the menu item's click event. If no path is specified, the click sound will not be used. If the path is incorrect, the click sound will not be used. The default value is undefined.

*volume* A number from 0 to 100 representing a volume level. 100 is full volume and 0 is no volume. The default setting is 100.

**Sound onRollover:** An object type parameter that contains two values to be set: source and volume.

*source* This parameter is the path to an external mp3 file loaded into a movie clip and attached to the menu item's rollover event. If no path is specified, the rollover sound will not be used. If the path is incorrect, the rollover sound will not be used. The default value is undefined.

*volume* A number from 0 to 100 representing a volume level. 100 is full volume and 0 is no volume. The default setting is 100.

**Top Level Icons:** Determines whether to draw the arrow icons for the top level menu items. The default value is "true".

*Note:* If both Top Level Icons and `Item Indent (level 2)` parameters are set to "false", then neither tree icons nor dotted lines for second level sub-menus are drawn, and each second level menu item (if it has nested third level sub-menu) stays open permanently.

**Tree Structure:** Specifies the appearance and initial state of Accordion Tree Menu structure. This parameter can be one of three predefined values: "open", "close", "static". The default value is "close".

*open* Each second level menu item that has nested third level sub-menu is in open state, and can be closed by clicking on appropriate tree icon.

*close* Each second level menu item that has nested third level sub-menu is in closed state, and can be opened by clicking on appropriate tree icon.

*static* Each second level menu item that has nested third level sub-menu stays open permanently. No tree icons are drawn.

*Note:* The component's structure appearance is also depends on values combination of the following parameters: Top Level Icons, Item Indent (level 2) and Dotted Line (level 2).

**Tweening Duration:** A number indicating the duration of the tweened animation in seconds. The tweened animation is applied to each third level sub-menu during its movement up and down. This parameter accepts values from 0.05 to 0.5. The default value is 0.3.

**XML-file URL:** This parameter is the path to configuration XML file. The default value is undefined.

*Note:* Another way to set the path to configuration XML file is to pass the variable `xmlURL` to a SWF (containing Accordion Tree Menu component instance) through HTML tags (see [example](#) for more information).

This is the basic template of your configuration XML file:

```

<?xml version="1.0" encoding="UTF-8"?>
<menu width="200">
  <properties>
    <treeStructure state="open"/>
    <itemHeight level1="24" level2="20"/>
    <topLevelIcons enabled="true"/>
    <dottedLine level2="true" level3="true"/>
    <decoration height="5" rounded="right"/>
    <leftMargin value=""/>
    <itemIndent level2="true" level3="true"/>
    <itemSeparators level1="true" level2="false"/>
    <tween duration="0.3" easing="regular" method="easeOut"/>
    <rolloverSound src="" volume=""/>
    <clickSound src="hifi.mp3" volume="80"/>
    <initialStateReturn enabled="true"/>
    <selectedStateMode enabled="true"/>
    <internalSelectionMode enabled="false"/>
  </properties>
  <style>
    <mainBackground color="F5F9FA" alpha="30"/>
    <decorationBackground color="E8EEF0" alpha=""/>
    <topLevelBackground color="E8EEF0" alpha=""/>
    <treelcon color="" bgcolor="F5F9FA"/>
    <dottedLine color=""/>
    <separatorLine color1="CCCCCC" color2="FFFFFF"/>
    <rolloverBackgroundColor level1="F5F9FA" level2="E8EEF0"/>
    <rolloverBackgroundAlpha level1="" level2="60"/>
    <selectedBackgroundColor level2="E8EEF0"/>
    <selectedBackgroundAlpha level2="60"/>
    <fontSize level1="14" level2="12" level3="11"/>
    <fontFamily name="Arial"/>
    <fontColor level1="000033" level2="000033" level3="000033"/>
    <fontColorRollover level1="CC3300" level2="CC3300" level3="CC3300"/>
    <fontColorSelected level2="CC3300" level3="CC3300"/>
    <fontWeight level1="bold" level2="" level3=""/>
    <fontWeightRollover level1="bold" level2="" level3=""/>
    <fontWeightSelected level2="" level3=""/>
    <fontStyle level1="" level2="" level3=""/>
    <textDecorationRollover level1="" level2="" level3=""/>
  </style>
  <items>
    <item label="Europe" action="getUrl" url="/world/europe/" target="_self" font_color="">
      <item label="Northern Europe" action="" url="" target="" font_color="">
        <item label="Norway" action="showMap" url="map3" target="_level0.maps"
          font_color=""/>
      </item>
    </item>
    .....
  </items>
</menu>

```

*Note:* Setting a property in the XML file overrides the parameter of the same name set in the Component inspector. If the value of a property in the XML file is undefined (omitted), it will be ignored.

The menu element has one optional attribute width which sets the width of an Accordion Tree Menu component instance.

The properties element determines mostly how Accordion Tree Menu component behaves. The following table describes the attributes of nodes that the properties element contains:

Node name	Attribute name	Default	Description
treeStructure	state	"close"	Specifies the appearance and initial state of Accordion Tree Menu structure. Possible values: "open", "close", "static".
itemHeight	level1	24	Indicates the height of each top level item, in pixels.
	level2	20	Indicates the height of each second and third level item, in pixels.
topLevelIcons	enabled	"true"	Determines whether to draw the arrow icons for the top level menu items. Possible values: "true", "false".
dottedLine	level2	"true"	Indicates whether the dotted lines for second level sub-menus to be drawn. Possible values: "true", "false".
	level3	"true"	Indicates whether the dotted lines for third level sub-menus to be drawn. Possible values: "true", "false".
decoration	height	0	Indicates the height of the top and bottom decorations, in pixels. Possible values: from 0 to the value of itemHeight (level 1).
	rounded	"all"	Specifies which corners of the menu decoration to be drawn rounded. Possible values: "left", "right", "all", "none".
leftMargin	value	0	Indicates the left margin of the menu, in pixels. This parameter accepts values from 0 to 100.
itemIndent	level2	"true"	Indicates whether to indent the second level items. Possible values: "true", "false".
	level3	"true"	Indicates whether to indent the third level items. Possible values: "true", "false".
itemSeparators	level1	"true"	Determines whether to draw separator lines between the top level menu items. Possible values: "true", "false".
	level2	"false"	Determines whether to draw separator lines between the second level menu items and between the third level menu items. Possible values: "true", "false".

tween	duration	0.3	A number indicating the duration of the tweened animation in seconds. The tweened animation is applied to each third level sub-menu during its movement up and down. Possible values: from 0.05 to 0.5.
	easing	"regular"	Specifies which easing class to be applied when opening or closing a second level menu item. Possible values: "regular", "none".
	method	"easeOut"	Specifies which easing method to be applied when opening or closing a second level menu item. Possible values: "easeIn", "easeOut", "easeNone".
rolloverSound	src	undefined	The path to an external mp3 file loaded into a movie clip and attached to the menu item's rollover event.
	volume	100	A number representing a volume level. Possible values: from 0 to 100.
clickSound	src	undefined	The path to an external mp3 file loaded into a movie clip and attached to the menu item's click event.
	volume	100	A number representing a volume level. Possible values: from 0 to 100.
initialStateReturn	enabled	"true"	Instructs Accordion Tree Menu how to behave when the mouse pointer rolls out of an open top level item outside the menu area. Possible values: "true", "false".
selectedStateMode	enabled	"true"	Turns on/off the component's ability to make an item selected. Possible values: "true", "false".
internalSelectionMode	enabled	"false"	Turns on/off the component's ability to make an item selected when it is clicked. That means Accordion Tree Menu can change the value of selectedIndex property inside a movie clip without reloading HTML page. Possible values: "true", "false".

The style element determines the appearance of Accordion Tree Menu component. The following table describes the attributes of nodes that the style element contains:

Node name	Attribute name	Default	Description
mainBackground	color	"FFFFFF"	The main background color of the menu. If omitted, the background is transparent.
	alpha	100	The main background alpha of the menu. Possible values: from 0 to 100.

decorationBackground	color	"FFFFFF"	The background color of the top and bottom decorations. If omitted, the background is transparent.
	alpha	100	The background alpha of the top and bottom decorations. Possible values: from 0 to 100.
topLevelBackground	color	"FFFFFF"	The background color of the top level menu items. If omitted, the background is transparent.
	alpha	100	The background alpha of the top level menu items. Possible values: from 0 to 100.
treelcon	color	–	The color of the tree icons. If omitted, the color is the same as the text color of a top level item.
	bgcolor	–	The background color of the tree icons. If omitted, the background color is the same as the main background color of the menu.
dottedLine	color	–	The color of the dotted lines. If omitted, the color is the same as the text color of a top level item.
separatorLine	color1	undefined	The color of the top separator line. If omitted, the line will not be drawn.
	color2	undefined	The color of the bottom separator line. If omitted, the line will not be drawn.
rolloverBackgroundColor	level1	"FFFFFF"	The background color of a rolled-over top level item. If omitted, the background is transparent.
	level2	"FFFFFF"	The background color of a rolled-over second or third level item. If omitted, the background is transparent.
rolloverBackgroundAlpha	level1	100	The background alpha of a rolled-over top level item. Possible values: from 0 to 100.
	level2	100	The background alpha of a rolled-over second or third level item. Possible values: from 0 to 100.
selectedBackgroundColor	level2	"FFFFFF"	The background color of a selected second or third level item. If omitted, the background is transparent.
selectedBackgroundAlpha	level2	100	The background alpha of a selected second or third level item. Possible values: from 0 to 100.

fontSize	level1	14	The point size for the font of a top level item text. Possible values: from 9 to 30.
	level2	12	The point size for the font of a second level item text. Possible values: from 9 to 22.
	level3	11	The point size for the font of a third level item text. Possible values: from 9 to 22.
fontFamily	name	"_sans"	The font name for text.
fontColor	level1	"000033"	The text color of a top level item.
	level2	"000033"	The text color of a second level item.
	level3	"000033"	The text color of a third level item.
fontColorRollover	level1	"CC3300"	The color of text when the pointer rolls over a top level item.
	level2	"CC3300"	The color of text when the pointer rolls over a second level item.
	level3	"CC3300"	The color of text when the pointer rolls over a third level item.
fontColorSelected	level2	"CC3300"	The color of text in the selected second level item.
	level3	"CC3300"	The color of text in the selected third level item.
fontWeight	level1	"none"	The font weight for the text of a top level item. Possible values: "none", "bold".
	level2	"none"	The font weight for the text of a second level item. Possible values: "none", "bold".
	level3	"none"	The font weight for the text of a third level item. Possible values: "none", "bold".
fontWeightRollover	level1	"none"	The font weight for the text when the pointer rolls over a top level item. Possible values: "none", "bold".
	level2	"none"	The font weight for the text when the pointer rolls over a second level item. Possible values: "none", "bold".
	level3	"none"	The font weight for the text when the pointer rolls over a third level item. Possible values: "none", "bold".
fontWeightSelected	level2	"none"	The font weight for the text in the selected second level item. Possible values: "none", "bold".
	level3	"none"	The font weight for the text in the selected third level item. Possible values: "none", "bold".

fontStyle	level1	"normal"	The font style for the text of a top level item. Possible values: "normal", "italic".
	level2	"normal"	The font style for the text of a second level item. Possible values: "none", "bold".
	level3	"normal"	The font style for the text of a third level item. Possible values: "none", "bold".
textDecorationRollover	level1	"none"	The text decoration when the pointer rolls over a top level item. Possible values: "none", "underline".
	level2	"none"	The text decoration when the pointer rolls over a second level item. Possible values: "none", "underline".
	level3	"none"	The text decoration when the pointer rolls over a third level item. Possible values: "none", "underline".

*Note:* All colors should be expressed in RRGGBB format.

The items element can contain an unlimited number of item nodes. Each item node represents a menu item object and should have one required attribute label. You can also add four optional attributes in the item node: action, url, target, font\_color.

Node name	Attribute name	Default	Description
item	label	undefined	The text that is displayed to represent a menu item.
	action	undefined	(Optional) This attribute can be either "getUrl" or the name of your custom Flash function (e.g., "showMap"). In the first case, Flash getURL() method is called — it loads a document from the specified URL into the specified window. In the second case, your custom Flash function is called.
	url	undefined	(Optional) This attribute defines either the URL from which to obtain the document (if action attribute is set to "getUrl") or any parameters to be passed to your function (if action attribute is set to the name of your custom Flash function). You can specify zero or more parameters, separating them by commas.
	target	"_self"	(Optional) This attribute can be either a parameter ("_self", "_blank", "_parent", "_top" or your custom value) that specifies the window or HTML frame that the document is loaded into (if action attribute is set to "getUrl") or a target movie clip where your function is placed (if action attribute is set to the name of your custom Flash function).
	font_color	undefined	(Optional) The text color of a single item (overrides the color setting for appropriate level in the style element).

## Sizing Accordion Tree Menu component

You can transform an Accordion Tree Menu component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `menuWidth` property.

The width of the bounding box of an Accordion Tree Menu component designates the width of the hit area for the component. If you increase the width of the component, you also increase the width of the hit area. The default value for a component's size is 100 x 100 pixels. If the component's bounding box is too small to fit the component's labels and graphic elements, they are clipped to fit.

Transforming a component instance vertically has no effect on its presentation after publishing (the value of the component's height is controlled programmatically). Transforming a component instance horizontally reduces or increases its width and does not resize the graphic elements and item labels of the menu. Change the component's width to fit the component labels if necessary. Modifying the component does not change the height of labels text.

## Creating an application with Accordion Tree Menu component

The following procedure explains how to add Accordion Tree Menu component to an application while authoring.

### Example 1

In this example, Accordion Tree Menu is used to create the navigation system for HTML-based web application.

To create an application with the Accordion Tree Menu component:

1. Set the Stage size to 120 x 580 pixels and specify "#89A7B1" as background color. For frame rate enter the value not less than 30 fps.
2. Drag one Accordion Tree Menu component from the Components panel to the Stage and set its coordinates to (0, 0).
3. In the Component inspector leave all parameters without changes (with default values).
4. Set dimensions for the component instances to 120 x 580 pixels.
5. Publish the Flash document with appropriate name — for example, "menu.swf".
6. Using your favorite text editor, create a new document and enter the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<menu width="">
  <properties>
    <treeStructure state="close"/>
    <itemHeight level1="20" level2="20"/>
    <topLevelIcons enabled="false"/>
    <dottedLine level2="true" level3="false"/>
    <decoration height="2" rounded="none"/>
    <leftMargin value="10"/>
    <itemIndent level2="true" level3="false"/>
    <itemSeparators level1="true" level2="false"/>
    <tween duration="" easing="" method=""/>
    <rolloverSound src="" volume=""/>
    <clickSound src="" volume=""/>
    <initialStateReturn enabled="true"/>
  </properties>
</menu>
```

```

        <selectedStateMode enabled="true"/>
        <internalSelectionMode enabled="false"/>
    </properties>
    <style>
        <mainBackground color="E8EEF0" alpha=""/>
        <decorationBackground color="00202B" alpha=""/>
        <topLevelBackground color="C5D4D9" alpha=""/>
        <treeIcon color="" bgcolor=""/>
        <dottedLine color=""/>
        <separatorLine color1="004B64" color2=""/>
        <rolloverBackgroundColor level1="C5D4D9" level2="C5D4D9"/>
        <rolloverBackgroundAlpha level1="" level2="30"/>
        <selectedBackgroundColor level2=""/>
        <selectedBackgroundAlpha level2=""/>
        <fontSize level1="13" level2="12" level3="11"/>
        <fontFamily name="Verdana"/>
        <fontColor level1="00202B" level2="00202B" level3="004B64"/>
        <fontColorRollover level1="00202B" level2="00202B" level3="004B64"/>
        <fontColorSelected level2="FF3300" level3="FF3300"/>
        <fontWeight level1="" level2="" level3=""/>
        <fontWeightRollover level1="" level2="" level3=""/>
        <fontWeightSelected level2="" level3=""/>
        <fontStyle level1="" level2="" level3=""/>
        <textDecorationRollover level1="" level2="" level3=""/>
    </style>
    <items>
        <item label="Austria" action="getUrl" url="destinations.php?index=0" target="">
            <item label="Vienna" action="getUrl" url="destinations.php?index=1" target="">
                <item label="Overview" action="getUrl" url="destinations.php?index=2" target=""/>
                <item label="Places to See" action="getUrl" url="destinations.php?index=3" target=""/>
                <item label="Money" action="getUrl" url="destinations.php?index=4" target=""/>
                <item label="Transport" action="getUrl" url="destinations.php?index=5" target=""/>
                <item label="Culture" action="getUrl" url="destinations.php?index=6" target=""/>
            </item>
            .....
        </item>
        .....
    </items>
</menu>

```

**Note:** In our case, the url attribute of item nodes refers to the same PHP-file — destinations.php. The index variable passed with GET method is used to make the necessary item of Accordion Tree Menu selected.

This is your configuration XML document. Save the XML file in preferred location (for example, in the same directory as the SWF file that contains an Accordion Tree Menu component) and name it, for example, "menu.xml".

7. Create a PHP file and place the following HTML code into it — to embed SWF file (in our case, menu.swf):

```

<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=8,0,0,0"
width="120" height="580" id="menu" align="middle">
    <param name="allowScriptAccess" value="sameDomain"/>
    <param name="movie" value="menu.swf"/>
    <param name="FlashVars" value="xmlURL=menu.xml&selIndex=<?=$index;?>"/>
    <param name="quality" value="high"/>

```

```

<param name="bgcolor" value="#89A7B1"/>
<embed src="menu.swf" FlashVars="xmlURL=menu.xml&selIndex=<?=$index;?>" quality="high"
bgcolor="#89A7B1" width="120" height="580" name="menu" align="middle"
allowScriptAccess="sameDomain" type="application/x-shockwave-flash"
pluginspage="http://www.adobe.com/go/getflashplayer"/>
</object>

```

*Note:* Accordion Tree Menu can accept two external variables: 1) xmlURL — the path to configuration XML file, 2) selIndex — the zero-based index of a menu item to be selected. In our case, selIndex is equal to the value of index variable passed with GET method in url.

8. Save the PHP file in preferred location (for example, in the same directory as the SWF file that contains an Accordion Tree Menu component) and name it "destinations.php".
9. Open destinations.php in a web browser and test the behavior of the Accordion Tree Menu component.

## Example 2

In this example, Accordion Tree Menu is used to create the navigation system for Flash-based web application.

Before creating an application, download the images required for this example from the following URLs and save them to your hard disk:

[http://www.e-merald.com/pic/components/car\\_bg.jpg](http://www.e-merald.com/pic/components/car_bg.jpg)

<http://www.e-merald.com/pic/components/beep.mp3>

To create an application with the Accordion Tree Menu component:

1. Set the Stage size to 520 x 500 pixels and specify "#000000" as background color. For frame rate enter the value not less than 30 fps.
2. Import the saved image file (car\_bg.jpg) to the Stage by selecting File > Import > Import to Stage. Set the coordinates for the imported image to (0, 0).
3. Drag one Accordion Tree Menu component from the Components panel to the Stage and set its coordinates to (20, 130). In the Property inspector, set the Instance Name value for the component instance to "atm".
4. In the Component inspector, do the following:  
For XML-file URL parameter, set: ..... "menu.xml"  
Leave the other parameters without changes.
5. Set dimensions for the component instance to 100 x 200 pixels.
6. Using the Text tool, create a dynamic text field on the Stage. With the text field selected, in the Property inspector, set the Variable value to "page\_title". Create two more dynamic text fields and set the Variable value for them accordingly to "section\_title" and "page\_text". Set appropriate values for other text field properties.
7. Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```

menuListener = new Object();
menuListener.onLoadXML = function(eventObj) {
    page_title = "Overview";
    page_text = "Description for this section...";
}

```

```

atm.addEventListener("onLoadXML", menuListener);
menuListener.click = function(eventObj) {
    if (eventObj.data == 0) atm.setItemSelected(1);
    if (eventObj.data == 5) atm.setItemSelected(6);
    if (eventObj.data == 11) atm.setItemSelected(12);
    if (eventObj.data == 14) atm.setItemSelected(15);
    section_title = atm.getItemAt(atm.selectedIndex).label;
    page_text = "Description for this section...";
}
atm.addEventListener("click", menuListener);
function pageTitle(topLevelIndex:Number):Void {
    page_title = atm.getItemAt(topLevelIndex).label.substr(0,1) +
    atm.getItemAt(topLevelIndex).label.substr(1).toLowerCase();
}

```

8. Publish the Flash document with appropriate name — for example, "menu.swf".
9. Using your favorite text editor, create a new document and enter the following code:

```

<?xml version="1.0" encoding="UTF-8"?>
<menu width="">
  <properties>
    <treeStructure state="static"/>
    <itemHeight level1="22" level2="18"/>
    <topLevelIcons enabled="true"/>
    <dottedLine level2="false" level3="false"/>
    <decoration height="3" rounded="none"/>
    <leftMargin value=""/>
    <itemIndent level2="false" level3=""/>
    <itemSeparators level1="false" level2="false"/>
    <tween duration="" easing="" method=""/>
    <rolloverSound src="" volume=""/>
    <clickSound src="beep.mp3" volume="70"/>
    <initialStateReturn enabled="true"/>
    <selectedStateMode enabled="true"/>
    <internalSelectionMode enabled="true"/>
  </properties>
  <style>
    <mainBackground color="" alpha=""/>
    <decorationBackground color="FFFFFF" alpha="30"/>
    <topLevelBackground color="FFFFFF" alpha="15"/>
    <treeIcon color="" bgcolor=""/>
    <dottedLine color=""/>
    <separatorLine color1="FFFFFF" color2=""/>
    <rolloverBackgroundColor level1="FFFFFF" level2=""/>
    <rolloverBackgroundAlpha level1="15" level2=""/>
    <selectedBackgroundColor level2=""/>
    <selectedBackgroundAlpha level2=""/>
    <fontSize level1="11" level2="10" level3=""/>
    <fontFamily name="Tahoma"/>
    <fontColor level1="CCCCCC" level2="CCCCCC" level3=""/>
    <fontColorRollover level1="CCCCCC" level2="FFB13E" level3=""/>
    <fontColorSelected level2="FFB13E" level3=""/>
    <fontWeight level1="bold" level2="" level3=""/>
    <fontWeightRollover level1="bold" level2="" level3=""/>
    <fontWeightSelected level2="" level3=""/>
    <fontStyle level1="" level2="" level3=""/>
    <textDecorationRollover level1="" level2="" level3=""/>
  </style>
</menu>

```

```

<items>
  <item label="FEATURES" action="pageTitle" url="0" target="" font_color="">
    <item label="Exterior" action="pageTitle" url="0" target="" font_color=""/>
    <item label="Interior" action="pageTitle" url="0" target="" font_color=""/>
    <item label="Safety" action="pageTitle" url="0" target="" font_color=""/>
    <item label="Price" action="pageTitle" url="0" target="" font_color=""/>
  </item>
  <item label="SPECS" action="pageTitle" url="5" target="" font_color="">
    <item label="Mechanical" action="pageTitle" url="5" target="" font_color=""/>
    <item label="Dimensions" action="pageTitle" url="5" target="" font_color=""/>
    <item label="Capacities" action="pageTitle" url="5" target="" font_color=""/>
    <item label="Tires" action="pageTitle" url="5" target="" font_color=""/>
    <item label="MPG" action="pageTitle" url="5" target="" font_color=""/>
  </item>
  <item label="OPTIONS" action="pageTitle" url="11" target="" font_color="">
    <item label="Available options" action="pageTitle" url="11" target="" font_color=""/>
    <item label="Packages" action="pageTitle" url="11" target="" font_color=""/>
  </item>
  <item label="ACCESSORIES" action="pageTitle" url="14" target="" font_color="">
    <item label="Exterior" action="pageTitle" url="14" target="" font_color=""/>
    <item label="Interior" action="pageTitle" url="14" target="" font_color=""/>
  </item>
</items>
</menu>

```

**Note:** In our case, the action attribute of each item node is equal to the name of Flash function to be called on menu item's click event. The url attribute of each item node is equal to the index of corresponding top level menu item. Thus, the value of url is a parameter for the pageTitle() function (see the ActionScript code above).

This is your configuration XML document. Save the XML file in preferred location (for example, in the same directory as the SWF file that contains an Accordion Tree Menu component) and name it, in our case, "menu.xml".

10. Open the Flash document (menu.swf).

## AccordionTreeMenu class

**Inheritance** MovieClip > UIObject class > AccordionTreeMenu class

Setting a property of the AccordionTreeMenu class with ActionScript overrides the parameter of the same name set in the Component inspector.

Setting a property in the XML file overrides that of the same name set with ActionScript (if the later has been applied not at run time).

### Method summary for the AccordionTreeMenu class

The following table lists methods of the AccordionTreeMenu class.

Method	Description
<a href="#">AccordionTreeMenu.getItemAt()</a>	Gets a reference to a menu item at a specified location.
<a href="#">AccordionTreeMenu.setItemSelected()</a>	Determines the index position of the menu item to be selected and changes the current selection.

### Methods inherited from the UIObject class

The following table lists the methods the AccordionTreeMenu class inherits from the UIObject class. When calling these methods from the AccordionTreeMenu object, use the form *AccordionTreeMenu.methodName*.

Method	Description
<a href="#">UIObject.getStyle()</a>	Gets the style property from the style declaration or object.
<a href="#">UIObject.move()</a>	Moves the object to the requested position.

### Property summary for the AccordionTreeMenu class

The following table lists properties of the AccordionTreeMenu class.

Property	Description
<a href="#">AccordionTreeMenu.decorationHeight</a>	The height of the top and bottom decorations, in pixels.
<a href="#">AccordionTreeMenu.decorationRounded</a>	Specifies which corners of the menu decoration to be drawn rounded.
<a href="#">AccordionTreeMenu.dottedLineL2</a>	Indicates whether the dotted lines for second level sub-menus to be drawn.
<a href="#">AccordionTreeMenu.dottedLineL3</a>	Indicates whether the dotted lines for third level sub-menus to be drawn.
<a href="#">AccordionTreeMenu.easingClass</a>	Specifies which easing class to be applied when opening or closing a second level menu item.
<a href="#">AccordionTreeMenu.easingMethod</a>	Specifies which easing method to be applied when opening or closing a second level menu item.
<a href="#">AccordionTreeMenu.initialStateReturn</a>	Instructs the menu how to behave when the mouse pointer rolls out of an open top level item outside the menu area.

<code>AccordionTreeMenu.internalSelectionMode</code>	Turns on/off the component's ability to make an item selected when it is clicked. That means the Accordion Tree Menu can change the value of <code>selectedIndex</code> property inside a movie clip without reloading HTML page.
<code>AccordionTreeMenu.itemHeightL1</code>	The height of each top level item, in pixels.
<code>AccordionTreeMenu.itemHeightL2</code>	The height of each second and third level item, in pixels.
<code>AccordionTreeMenu.itemIndentL2</code>	Indicates whether to indent the second level items.
<code>AccordionTreeMenu.itemIndentL3</code>	Indicates whether to indent the third level items.
<code>AccordionTreeMenu.itemSeparatorsL1</code>	Determines whether to draw separator lines between the top level menu items.
<code>AccordionTreeMenu.itemSeparatorsL2</code>	Determines whether to draw separator lines between the second level menu items and between the third level menu items.
<code>AccordionTreeMenu.leftMargin</code>	The left margin of the menu, in pixels.
<code>AccordionTreeMenu.menuWidth</code>	The width of the menu, in pixels.
<code>AccordionTreeMenu.selectedIndex</code>	Read-only; the index position of the selected menu item.
<code>AccordionTreeMenu.selectedStateMode</code>	Turns on/off the component's ability to make an item selected.
<code>AccordionTreeMenu.topLevelIcons</code>	Determines whether to draw the arrow icons for the top level menu items.
<code>AccordionTreeMenu.treeStructure</code>	Specifies the appearance and initial state of Accordion Tree Menu structure.
<code>AccordionTreeMenu.tweeningDuration</code>	A number indicating the duration of the tweened animation in seconds. The tweened animation is applied to each third level sub-menu during its movement up and down.

### Properties inherited from the UIObject class

The following table lists the properties the `AccordionTreeMenu` class inherits from the `UIObject` class. When accessing these properties from the `AccordionTreeMenu` object, use the form *AccordionTreeMenu.propertyName*.

Property	Description
<code>UIObject.left</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.top</code>	Read-only; the position of the top edge of the object, relative to its parent.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (true) or not (false).
<code>UIObject.x</code>	Read-only; the left edge of the object, in pixels.
<code>UIObject.y</code>	Read-only; the top edge of the object, in pixels.

## Event summary for the `AccordionTreeMenu` class

The following table lists the event of the `AccordionTreeMenu` class.

Event	Description
<code>AccordionTreeMenu.click</code>	Broadcast when a menu item is clicked.
<code>AccordionTreeMenu.closing</code>	Broadcast when a top level item is closing.
<code>AccordionTreeMenu.onLoadXML</code>	Broadcast when the component attempts to load an XML file.
<code>AccordionTreeMenu.opening</code>	Broadcast when a top level item is opening.
<code>AccordionTreeMenu.rollOut</code>	Broadcast when the pointer rolls off a menu item.
<code>AccordionTreeMenu.rollOver</code>	Broadcast when the pointer rolls over a menu item.

## Events inherited from the `UIObject` class

The following table lists the events the `AccordionTreeMenu` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.

## `AccordionTreeMenu.click`

### Usage

Usage 1:

```
listenerObject = new Object();  
listenerObject.click = function(eventObject) {  
    // Your code here  
}  
my_menu.addEventListener("click", listenerObject);
```

Usage 2:

```
on (click) {  
    // Your code here  
}
```

### Description

Event; broadcast to all registered listeners when the mouse is clicked (pressed) over a menu item.

The first usage example uses a dispatcher/listener event model. A component instance (*my\_menu*) dispatches an event (in this case, `click`) and the event is handled by a function, also called a "handler", on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

The second usage example uses an `on()` handler and must be attached directly to a menu instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component

instance. For example, the following code, attached to a Accordion Tree Menu instance *my\_menu*, sends "\_level0.my\_menu" to the Output panel:

```
on (click) {
    trace(this);
}
```

### Event object

Along with the standard event object properties, the click event has one additional property — an integer that indicates the index position of the clicked menu item.

### Example

This example, written on a frame of the Timeline, sets the `selected_item` variable to the value of the clicked menu item's "label" attribute. The target property of an event object is the component that generated the event. You can access instance properties and methods from the target property (in this example, the `getItemAt()` method is accessed).

```
var selected_item:String;
menuListener = new Object();
menuListener.click = function(eventObj) {
    selected_item = eventObj.target.getItemAt(eventObj.data).label;
}
my_menu.addEventListener("click", menuListener);
```

The following code sends a message to the Output panel when a menu item is clicked. The `on()` handler must be attached directly to a menu instance.

```
on (click) {
    trace("menu item was clicked");
}
```

## AccordionTreeMenu.closing

### Usage

Usage 1:

```
listenerObject = new Object();
listenerObject.closing = function(eventObject) {
    // Your code here
}
my_menu.addEventListener("closing", listenerObject);
```

Usage 2:

```
on (closing) {
    // Your code here
}
```

### Description

Event; broadcast to all registered listeners when a top level item of the Accordion Tree Menu component is closing.

The first usage example uses a dispatcher/listener event model. A component instance (*my\_menu*) dispatches an event (in this case, `closing`) and the event is handled by a function, also called a "handler", on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the

`EventDispatcher.addListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

The second usage example uses an `on()` handler and must be attached directly to a menu instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to a `Accordion Tree Menu` instance `my_menu`, sends `"_level0.my_menu"` to the Output panel:

```
on (closing) {  
    trace(this);  
}
```

### Event object

Along with the standard event object properties, the closing event has one additional property that can retrieve the following values:

- `start` (String) Triggers when the nested second level sub-menu starts moving up.
- `moving` (String) Triggers continuously while the nested second level sub-menu keeps moving up.
- `stop` (String) Triggers when the nested second level sub-menu stops moving up.

### Example

This example, written on a frame of the Timeline, sends messages to the Output panel when the top level item is closing.

```
menuListener = new Object();  
menuListener.closing = function(eventObj) {  
    if (eventObj.data == "start") trace("menu item starts closing");  
    if (eventObj.data == "moving") trace("menu item keeps moving");  
    if (eventObj.data == "stop") trace("menu item stops closing");  
}  
my_menu.addListener("closing", menuListener);
```

The following code sends a message to the Output panel when a top level item is closing. The `on()` handler must be attached directly to a menu instance.

```
on (closing) {  
    trace("menu item is closing");  
}
```

## AccordionTreeMenu.decorationHeight

### Usage

```
my_menu.decorationHeight
```

### Description

Property; the height, in pixels, of the top and bottom decorations. Possible values: from 0 to the value of a top level item height. If `decorationHeight` is equal to 0, then no decoration will be drawn. The default value is 0.

### Example

The following example sets the `decorationHeight` property to 5 pixels:

```
my_menu.decorationHeight = 5;
```

## AccordionTreeMenu.decorationRounded

### Usage

```
my_menu.decorationRounded
```

### Description

Property; a string that specifies which corners of the menu decoration to be drawn rounded. This property can be one of four predefined values: "left", "right", "all", "none". The default value is "all".

### Example

The following example sets the decorationRounded property to "right":

```
my_menu.decorationRounded = "right";
```

## AccordionTreeMenu.dottedLineL2

### Usage

```
my_menu.dottedLineL2
```

### Description

Property; indicates whether the dotted lines for second level sub-menus to be drawn ("true") or not ("false"). If dottedLineL2 is set to "false", then no tree icons are drawn, and each second level menu item (if it has nested third level sub-menu) stays open permanently. The default value is "true".

### Example

The following example sets the dottedLineL2 property to "false":

```
my_menu.dottedLineL2 = false;
```

## AccordionTreeMenu.dottedLineL3

### Usage

```
my_menu.dottedLineL3
```

### Description

Property; indicates whether the dotted lines for third level sub-menus to be drawn ("true") or not ("false"). The default value is "true".

### Example

The following example sets the dottedLineL3 property to "false":

```
my_menu.dottedLineL3 = false;
```

## AccordionTreeMenu.easingClass

### Usage

```
my_menu.easingClass
```

### Description

Property; a string that specifies which easing class to be applied when opening or closing a second level menu item. This property can be either "regular" or "none". The default value is "regular".

### Example

The following example sets the easingClass property to "none":

```
my_menu.easingClass = "none";
```

## AccordionTreeMenu.easingMethod

### Usage

```
my_menu.easingMethod
```

### Description

Property; a string that specifies which easing method to be applied when opening or closing a second level menu item. This property can be one of three predefined values: "easeIn", "easeOut" or "easeNone"; the default value is "easeOut".

### Example

The following example sets the easingMethod property to "easeIn":

```
my_menu.easingMethod = "easeIn";
```

## AccordionTreeMenu.getItemAt()

### Usage

```
my_menu.getItemAt(index)
```

### Parameters

*index* An integer indicating the index of the node in the menu. This is a zero-based index, so 0 retrieves the first item, 1 retrieves the second item, and so on.

### Returns

A reference to the specified node.

### Description

Method; returns a reference to the specified child node of the menu.

### Example

The following example demonstrates how to retrieve the values of attributes available for an item node:

```
my_menu.getItemAt(0).label;  
my_menu.getItemAt(0).action;  
my_menu.getItemAt(0).url;  
my_menu.getItemAt(0).target;  
my_menu.getItemAt(0).fontColor;
```

## AccordionTreeMenu.initialStateReturn

### Usage

```
my_menu.initialStateReturn
```

### Description

Property; instructs the menu how to behave when the mouse pointer rolls out of an open top level item outside the menu area. There are two values available:

*true* The open top level item remains open.

*false* The menu returns to its initial state. If [selectedStateMode](#) is set to "true" and the [selectedIndex](#) property has a definite value, then the top level item including nested item, which index is equal to the selectedIndex value, will be opened. If there is no item selected, the open top level item will be closed.

The default value is "true".

## Example

The following example sets the `initialStateReturn` property to "false":

```
my_menu.initialStateReturn = false;
```

## AccordionTreeMenu.internalSelectionMode

### Usage

```
my_menu.internalSelectionMode
```

### Description

Property; turns on/off the component's ability ("true" or "false") to make an item selected when it is clicked. That means the Accordion Tree Menu can change the value of the `selectedIndex` property inside a movie clip without reloading HTML page. This option is useful, first of all, for Flash-based web applications. For this property to be activated, `selectedStateMode` has to be set to "true". The default value is "false".

### Example

The following example sets the `internalSelectionMode` property to "true":

```
my_menu.internalSelectionMode = true;
```

## AccordionTreeMenu.itemHeightL1

### Usage

```
my_menu.itemHeightL1
```

### Description

Property; the height, in pixels, of each top level item. The value of this property is controlled programmatically, considering specified font size and font family. The default value is 24.

### Example

The following example sets the `itemHeightL1` property to 30:

```
my_menu.itemHeightL1 = 30;
```

## AccordionTreeMenu.itemHeightL2

### Usage

```
my_menu.itemHeightL2
```

### Description

Property; the height, in pixels, of each second and third level item. The value of this property is controlled programmatically, considering specified font size and font family. The default value is 20.

### Example

The following example sets the `itemHeightL2` property to 24:

```
my_menu.itemHeightL2 = 24;
```

## AccordionTreeMenu.itemIndentL2

### Usage

```
my_menu.itemIndentL2
```

## Description

Property; indicates whether to indent the second level items. The amount of indent is determined programmatically. The default value is "true". If both `itemIndentL2` and `topLevelIcons` properties are set to "false", then neither tree icons nor dotted lines for second level sub-menus are drawn, and each second level menu item (if it has nested third level sub-menu) stays open permanently.

## Example

The following example sets the `itemIndentL2` property to "false":

```
my_menu.itemIndentL2 = false;
```

## AccordionTreeMenu.itemIndentL3

### Usage

```
my_menu.itemIndentL3
```

### Description

Property; indicates whether to indent the third level items. The amount of indent is determined programmatically. The default value is "true". If `itemIndentL3` is set to "false", then no dotted lines for third level sub-menus are drawn.

### Example

The following example sets the `itemIndentL3` property to "false":

```
my_menu.itemIndentL3 = false;
```

## AccordionTreeMenu.itemSeparatorsL1

### Usage

```
my_menu.itemSeparatorsL1
```

### Description

Property; determines whether to draw separator lines between the top level menu items. The default value is "true".

### Example

The following example sets the `itemSeparatorsL1` property to "false":

```
my_menu.itemSeparatorsL1 = false;
```

## AccordionTreeMenu.itemSeparatorsL2

### Usage

```
my_menu.itemSeparatorsL2
```

### Description

Property; determines whether to draw separator lines between the second level menu items and between the third level menu items. The default value is "false".

### Example

The following example sets the `itemSeparatorsL2` property to "true":

```
my_menu.itemSeparatorsL2 = true;
```

## AccordionTreeMenu.leftMargin

### Usage

```
my_menu.leftMargin
```

### Description

Property; the left margin of the menu, in pixels. Accepts values from 0 to 100. The default value is 0.

### Example

The following example sets the leftMargin property to 20:

```
my_menu.leftMargin = 20;
```

## AccordionTreeMenu.menuWidth

### Usage

```
my_menu.menuWidth
```

### Description

Property; the width of the menu, in pixels. The default value is 100.

### Example

The following example sets the menuWidth property to 200:

```
my_menu.menuWidth = 200;
```

## AccordionTreeMenu.onLoadXML

### Usage

Usage 1:

```
listenerObject = new Object();  
listenerObject.onLoadXML = function(eventObject) {  
    // Your code here  
}  
my_menu.addEventListener("onLoadXML", listenerObject);
```

### Description

Event; broadcast when the Accordion Tree Menu component attempts to load an XML file.

The usage example uses a dispatcher/listener event model. A component instance (*my\_menu*) dispatches an event (in this case, `onLoadXML`) and the event is handled by a function, also called a "handler", on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

### Event object

Contains a Boolean value ("true" or "false") to indicate whether an XML file was found.

## Example

This example, written on a frame of the Timeline, uses the event object to set up a conditional statement that checks to see if an XML file was found and loaded.

```
menuListener = new Object();
menuListener.onLoadXML = function(eventObj) {
    if (eventObj.data == true) {
        // show success alert
    } else {
        // show error alert
    }
}
my_menu.addEventListener("onLoadXML", menuListener);
```

## AccordionTreeMenu.opening

### Usage

Usage 1:

```
listenerObject = new Object();
listenerObject.opening = function(eventObject) {
    // Your code here
}
my_menu.addEventListener("opening", listenerObject);
```

Usage 2:

```
on (opening) {
    // Your code here
}
```

### Description

Event; broadcast to all registered listeners when a top level item of the Accordion Tree Menu component is opening.

The first usage example uses a dispatcher/listener event model. A component instance (*my\_menu*) dispatches an event (in this case, *opening*) and the event is handled by a function, also called a "handler", on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

The second usage example uses an `on()` handler and must be attached directly to a menu instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to a Accordion Tree Menu instance *my\_menu*, sends "`_level0.my_menu`" to the Output panel:

```
on (opening) {
    trace(this);
}
```

## Event object

Along with the standard event object properties, the opening event has one additional property that can retrieve the following values:

- *start* (String) Triggers when the nested second level sub-menu starts moving down.
- *moving* (String) Triggers continuously while the nested second level sub-menu keeps moving down.
- *stop* (String) Triggers when the nested second level sub-menu stops moving down.

## Example

This example, written on a frame of the Timeline, sends messages to the Output panel when the top level item is opening.

```
menuListener = new Object();
menuListener.opening = function(eventObj) {
    if (eventObj.data == "start") trace("menu item starts opening");
    if (eventObj.data == "moving") trace("menu item keeps moving");
    if (eventObj.data == "stop") trace("menu item stops opening");
}
my_menu.addEventListener("opening", menuListener);
```

The following code sends a message to the Output panel when a top level item is opening. The on() handler must be attached directly to a menu instance.

```
on (opening) {
    trace("menu item is opening");
}
```

## AccordionTreeMenu.rollOut

### Usage

Usage 1:

```
listenerObject = new Object();
listenerObject.rollOut = function(eventObject) {
    // Your code here
}
my_menu.addEventListener("rollOut", listenerObject);
```

Usage 2:

```
on (rollOut) {
    // Your code here
}
```

### Description

Event; broadcast to all registered listeners when the pointer rolls off a menu item.

The usage example uses a dispatcher/listener event model. A component instance (*my\_menu*) dispatches an event (in this case, *rollOut*) and the event is handled by a function, also called a "handler", on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

The second usage example uses an `on()` handler and must be attached directly to a menu instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to a `Accordion Tree Menu` instance `my_menu`, sends `"_level0.my_menu"` to the Output panel:

```
on (rollOut) {
    trace(this);
}
```

### Event object

Contains the following two properties:

*index* An integer that indicates the index position of the menu item that the pointer rolled off.

*area* A string that indicates the area to which the pointer rolled off. There are two possible values of this property: "inside", "outside".

### Example

This example, written on a frame of the Timeline, sets the `rollout_item` variable to the value of the "label" attribute of the menu item that the pointer rolled off. The `target` property of an event object is the component that generated the event. You can access instance properties and methods from the `target` property (in this example, the `getItemAt()` method is accessed).

```
var rollout_item:String;
menuListener = new Object();
menuListener.rollOut = function(eventObj) {
    rollout_item = eventObj.target.getItemAt(eventObj.data.index).label;
}
my_menu.addEventListener("rollOut", menuListener);
```

The following code sends a message to the Output panel when the pointer rolls off a menu item. The `on()` handler must be attached directly to a menu instance.

```
on (rollOut) {
    trace("the pointer rolls off a menu item");
}
```

## AccordionTreeMenu.rollOver

### Usage

Usage 1:

```
listenerObject = new Object();
listenerObject.rollOver = function(eventObject) {
    // Your code here
}
my_menu.addEventListener("rollOver", listenerObject);
```

Usage 2:

```
on (rollOver) {
    // Your code here
}
```

### Description

Event; broadcast to all registered listeners when the pointer rolls over a menu item.

The usage example uses a dispatcher/listener event model. A component instance (`my_menu`) dispatches an event (in this case, `rollOver`) and the event is handled by a function, also called a "handler", on a listener object (`listenerObject`) that you create. You define a method with the same

name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

The second usage example uses an `on()` handler and must be attached directly to a menu instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to a Accordion Tree Menu instance *my\_menu*, sends `"_level0.my_menu"` to the Output panel:

```
on (rollOver) {  
    trace(this);  
}
```

### Event object

Contains the following two properties:

*index*      An integer that indicates the index position of the menu item that the pointer rolled over.  
*area*        A string that indicates the area to which the pointer rolled over. There are two possible values of this property: "inside", "outside".

### Example

This example, written on a frame of the Timeline, sets the `rollover_item` variable to the value of the "label" attribute of the menu item that the pointer rolled over. The target property of an event object is the component that generated the event. You can access instance properties and methods from the target property (in this example, the `getItemAt()` method is accessed).

```
var rollover_item:String;  
menuListener = new Object();  
menuListener.rollOver = function(eventObj) {  
    rollover_item = eventObj.target.getItemAt(eventObj.data.index).label;  
}  
my_menu.addEventListener("rollOver", menuListener);
```

The following code sends a message to the Output panel when the pointer rolls over a menu item. The `on()` handler must be attached directly to a menu instance.

```
on (rollOver) {  
    trace("the pointer rolls over a menu item");  
}
```

## AccordionTreeMenu.selectedIndex

### Usage

```
my_menu.selectedIndex
```

### Description

Property (read-only); the index position of the selected menu item. The value is undefined if nothing is selected. The `selectedIndex` property has a definite (not undefined) value in the following three cases:

1. The variable `selIndex` is passed to a SWF (containing Accordion Tree Menu component instance) through HTML tags (see [example](#) for more information). If the component instance is not at the top level of a movie, you must specify the `selIndex` variable at the component's level.

2. The `internalSelectionMode` property is equal to "true" and a menu item has been clicked.
3. The `setItemSelected()` method has been called.

### Example

The following example gets the value of the `selectedIndex` property:

```
trace(my_menu.selectedIndex);
```

## AccordionTreeMenu.selectedStateMode

### Usage

```
my_menu.selectedStateMode
```

### Description

Property; turns on/off the component's ability ("true" or "false") to make an item selected if the `selectedIndex` property has a definite value. The default value is "true".

### Example

The following example sets the `selectedStateMode` property to "false":

```
my_menu.selectedStateMode = false;
```

## AccordionTreeMenu.setItemSelected()

### Usage

```
my_menu.setItemSelected(index)
```

### Parameters

*index* An integer indicating the zero-based index of the menu item to be selected. If this parameter is omitted, no item will be selected and the `selectedIndex` property will be set to the undefined value.

### Returns

Nothing.

### Description

Method; determines the index position of the menu item to be selected and changes the current selection. If you call `setItemSelected()` method, any current selection is cleared and the indicated item is selected.

### Example

The following example sets the index of the menu item to be selected to 10:

```
my_menu.setItemSelected(10);
```

## AccordionTreeMenu.topLevelIcons

### Usage

```
my_menu.topLevelIcons
```

### Description

Property; determines whether to draw the arrow icons for the top level menu items. If both `topLevelIcons` and `itemIndentL2` properties are set to "false", then neither tree icons nor dotted lines for second level sub-menus are drawn, and each second level menu item (if it has nested third level sub-menu) stays open permanently. The default value is "true".

## Example

The following example sets the `topLevelIcons` property to "false":

```
my_menu.topLevelIcons = false;
```

## AccordionTreeMenu.treeStructure

### Usage

```
my_menu.treeStructure
```

### Description

Property; specifies the appearance and initial state of Accordion Tree Menu structure. This parameter can be one of three predefined values:

*open* Each second level menu item that has nested third level sub-menu is in open state, and can be closed by clicking on appropriate tree icon.

*close* Each second level menu item that has nested third level sub-menu is in closed state, and can be opened by clicking on appropriate tree icon.

*static* Each second level menu item that has nested third level sub-menu stays open permanently. No tree icons are drawn.

The component's structure appearance is also depends on values combination of the following properties: `topLevelIcons`, `itemIndentL2` and `dottedLineL2`. The default value is "close".

### Example

The following example sets the `treeStructure` property to "static":

```
my_menu.treeStructure = "static";
```

## AccordionTreeMenu.tweeningDuration

### Usage

```
my_menu.tweeningDuration
```

### Description

Property; a number indicating the duration of the tweened animation in seconds. The tweened animation is applied to each third level sub-menu during its movement up and down. This parameter accepts values from 0.05 to 0.5. The default value is 0.3.

### Example

The following example sets the `tweeningDuration` property to 0.5:

```
my_menu.tweeningDuration = 0.5;
```