



CorePlayer 1.xx

Universal Skins / CoreUI

Skinning Specification

Revision #4

Last modified on January 10, 2007

Written by Phillip Hendrickson, with help from;

- **Dan 'BetaBoy' Marlin**
- **Steve Lhomme**
- **Picard**

CoreCodec, Inc.

Table of contents

Introduction.....	1
Introduction to XML.....	1
CorePlayer variables & menu list (language file).....	2
Element syntax.....	3
Bitmap.....	4
Template.....	4
Application (APP).....	5
Style.....	7
Mode.....	8
Group.....	9
Image (IMG).....	11
Slider.....	12
Thumb.....	14
Window move control (WINMOVE).....	15
Button.....	16
Label.....	17
Window resize control (WINSIZE).....	18
Fill.....	19
Viewport.....	20
Enumbox.....	21
Popupmenu.....	22
Split.....	23
Splitbar.....	24
Splititem.....	24
Tabkey.....	25
Properties.....	26
Poly.....	26
Dialog.....	27
Menu.....	28
Shortcut (hotkey).....	30
Skin structure.....	31
File structure and names.....	32
Conclusion.....	33
Appendix A: CorePlayer variable reference.....	i
Appendix B: CorePlayer language file.....	vii

Introduction

The process of expressing one's creativity or designing and producing one's own unique work can bring an him great fulfillment and satisfaction, especially if his creation has special beauty, functionality or both. Enter CorePlayer Universal Skins.

The purpose behind CorePlayer Universal Skins (US) is to create a skinning UI platform that is as efficient, robust and expansive as the CorePlayer framework itself. In other words, the US framework allows you to create a single skin that supports all of the CorePlayer platforms. So if done properly, you can create a skin that works for CE, Windows Mobile, Pocket PC, Windows Smartphone, Palm, Symbian, Windows, Linux and OS X devices and desktops.

Additionally, Universal Skins are the front end to the CorePlayer Common UI or what we call CoreUI. CoreUI is an ultra lightweight widget that allows you to skin any component within the CorePlayer Framework. That means that you can skin menu items, options menus, the user interface, Equalizer, Database, Media Library, etc. There are no limitations to the type of skin you can create.

What does this all mean for you as a skin designer? It means that you are completely free to express your tastes, your needs, and perhaps to address those of others using CorePlayer. This also means that others will also be designing their own skins. As all who may be interested in the design process will not start off with the same experience and background as you or others have, criticism of their work will be frowned upon, while helpful tips and hints where asked for are encouraged.

What follows in the rest of this document is an introduction to XML for those who have no experience with it. Following that brief introduction comes the meat of the document: the types of objects that you, the skin designer, can use to create your unique skin, how you can fit them together to create your skin, and what the file structure of the skin should be.

Introduction to XML

What is XML? XML stands for Extensible Markup Language and is a way of both describing the structure of data in a document and of including that data in the same document. Thus it allows people who know the specification for the document structure to both write documents for others and read the documents that others produce. The elegant thing about XML is that it is simple and completely arbitrary, so an XML document can take almost any structure as long as those who will be reading the document know what the structure is.

The heart of an XML document is an *element*, which may have *attributes* and/or *content*. Part or all of an element's content can be composed of child elements, so complex element structures can be formed. The syntax for an element declaration is as follows:

```
<NAME ATTRIBUTE="value" attribute="value"... />
```

OR

```
<NAME ATTRIBUTE="value" attribute="value"...>
  content
</NAME>
```

You will notice that the name of the element follows an opening angle bracket (<) and that all attributes follow the element name in the initial declaration. Notice, too, that all attribute values must be enclosed in quotation marks. Then, if the element contains no content, a forward slash and closing angle bracket (/>) finish the declaration. In cases where content is present, the initial element declaration should be closed with an angle bracket (>), and the content then follows, indented, on a new line (or several, if there's a lot). To finish the declaration, </name> goes on the last line, and is not indented, where 'name' refers to the name of the element.

Note that this is only a very brief introduction to XML so that anyone interested in creating a CP skin has the chance. For more detailed information about XML, please see

<http://en.wikipedia.org/wiki/XML>.

CorePlayer variables & menu list (language file)

CorePlayer contains many internal variables that can be referenced by elements in the skin file. Examples of these variables include the following:

- | | |
|--------------------------|---|
| ○ player.play | -gets or sets a value determining whether CP is currently playing |
| ○ player.shuffle | -determines whether CP is in shuffle mode or not |
| ○ player.percent | -contains the percentage complete of the current file |
| ○ player.title | -contains the title of the current file |
| ○ equalizer.preamplifier | -gets or sets the current pre-amp level of the player |
| ○ audio.mute | -determines whether the sound is muted or not |

These variables are the way that controls can be mapped to certain aspects of the player. For example, a slider can be created that shows the percentage complete of the current media file, and the way that it updates itself is by periodically checking the value of `player.percent`. Note that the list above is only a very limited subset of the full list, which list can be found in Appendix A of this document.

Additionally, since English speakers aren't the only users of CorePlayer, menu items and other text-based components of the skin have to be translated into other languages. This is done by including them in a language file, and then referencing the entries for the language file in the skin definition. For example, suppose I have a menu item to close CorePlayer. Instead of setting the text of the menu item = "close", I set it = "#UI__0102", where the '#' tells CP to look up the text under the entry in the table labeled "UI__0102". That way, the skin designer doesn't have to create a separate skin for each language that CP supports, but instead only has to create a single skin that references the language file. For the complete English language file, see Appendix B of this document.

Element syntax

Since the universal skin definition for CorePlayer 1.xx is an XML document which is interpreted by a built-in XML parser, if you want to create your own skin for CP, you need to know how the document should be structured. In CP, there are several different kinds of XML elements, or classes, which do different things. Each of these classes also has various named elements that can be referenced by other sub-elements within the class. Below you will find a listing of those class types, their named elements, what they (the classes) do, common child elements they can contain, a listing of all possible attributes (each with a description of what valid values are for it), and an example of how to use the element in a skin.

First, though, I need to note two or three things:

1. All attributes of all classes have a default value of 0. Thus, if a class has an optional attribute that you don't need, or whose value is going to be 0 anyway, you can make your skin less cluttered by leaving that attribute out of the element declaration.
2. Each attribute requires a specific data type. These types include integers, strings, booleans, and CP function names, variables and events. If an attribute requires a boolean value, all of the normal C boolean operators can be used. These include '||' for an 'or' operation, '&&' for an 'and' operation, and '!' for a 'not' operation.

Now, on to the element / class descriptions!

Bitmap

A bitmap is an element that describes an image to be used with your CP skin. This is a very common element as a great number of the components of a skin are graphical, from buttons to scrollbars and backgrounds.

Named elements:

Content / child elements:

None. A BITMAP consists only of attributes..

Syntax:

```
<BITMAP [SKIP] NAME DATA [DPI] [MASK] />
```

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine. This is the most common way to differentiate between devices (and thus alter the skin for different devices), with common device options being the following:
 - platform.desktop – any desktop computer
 - platform.mouse – anything that has mouse input (a PC) or a touch-sensitive screen (most PDAs)
- **NAME:** Required. Data type -> string. Gives the BITMAP a unique name that distinguishes it from other elements in the file.
- **DATA:** Required. Data type -> file name. The name of the image file to be associated with this bitmap. The image must exist in the same directory or same .zip (.CPS) file as the skin file.
- **DPI:** Optional. Data type -> integer. Sets resolution of the image.
- **MASK:** Optional. Data type -> Hexadecimal integer (e.g. “#0000FF”). Sets the color that will be interpreted as transparent.

Example:

- `<BITMAP SKIP="!platform.desktop" NAME="close" DATA="close_small.bmp" />`

This example creates a bitmap named “close” using the “close_small.bmp” image file. It only creates the BITMAP element if the platform that CP is running on is a desktop computer.

Template

A template is an element that creates some portion of the CP skin and which can be reused repeated times in the skin. For example, suppose I want to create a menu bar which will be used in three different CP play modes (see MODE definition below). Rather than copy and paste the XML code to create that

menu bar in three different places, I can create a TEMPLATE for the menu bar and then just reference the template in each of the three places I want it.

Named elements:

None.

Content / child elements:

Any available class type except DIALOG, MENU, APP and HOTKEY.

Syntax:

<TEMPLATE NAME />

Attributes:

- **NAME:** Required. Data type -> string. Gives the TEMPLATE a unique name that distinguishes it from other elements in the file.

Example:

- **<TEMPLATE NAME="main_top">**
 (*content*)
</TEMPLATE>

This example creates a template named “main_top”. Anytime **<main_top />** gets written in the rest of the skin file, the (*content*) of the TEMPLATE will be pasted into that location.

Application (APP)

An APP is an element that represents the main CP program window, and contains the skin information for any modes that the skin designer specifies (see MODE description below). In any given skin file, there should be only a single APP object.

Named elements:

Content / child elements:

Any available class type except DIALOG, MENU and HOTKEY. For a description of how content should be organized within an APP element, see the next section (“Skin Structure”) of this document.

Syntax:

- **<APP NAME WIDTH HEIGHT [MINWIDTH] [MINHEIGHT] TASKBAR CAPTION SOFTMENU HOTKEY>**
 (*content*)

</APP>

Attributes:

- **NAME:** Required. Data type -> string. Gives the APP a unique name that distinguishes it from other elements in the file.
- **WIDTH:** Required. Data type -> integer. Specifies the width of the entire window, which either gets scaled or ignored if CP occupies the full screen (as is the case for most PDA's).
- **HEIGHT:** Required. Data type -> integer. Specifies the height of the entire window, which gets scaled depending on the screen size of the device.
- **MINWIDTH:** Optional. Data type -> integer. Specifies the smallest width the window can take. This is mostly for devices which have resizable windows (to keep the user from making the window arbitrarily small).
- **MINHEIGHT:** Optional. Data type -> integer. Specifies the smallest height the window can take. This is mostly for devices which have resizable windows (to keep the user from making the window arbitrarily small).
- **TASKBAR:** Required. Data type -> integer. Sets whether the taskbar is visible or not (0=no, 1=yes)
- **CAPTION:** Required. Data type -> integer. Sets whether CP has a caption or not (0=no, 1=yes)
- **SOFTMENU:** Required. Data type -> String. Sets the menu system for the main window & must be the name of a menu group defined somewhere in the skin definition – see MENU syntax definition below.
- **HOTKEY:** Required. Data type -> string. Sets the hotkey scheme for the main window & must be the name of a group of hotkeys defined somewhere in the skin definition – see HOTKEY syntax definition below.

Example:

- `<APP NAME="main" WIDTH="400" HEIGHT="400" MINWIDTH="240" MINHEIGHT="240" TASKBAR="1" CAPTION="0" SOFTMENU="main_softmenu" HOTKEY="main_hotkey">`
 - *(definitions here)*
- `</APP>`

This initializes an APP element with a window size of 400x400 pixels, with a taskbar, without a caption, with the “main_softmenu” menu and with the “main_hotkey” set of shortcuts. The 400x400 window size gets ignored on devices where CP runs maximized.

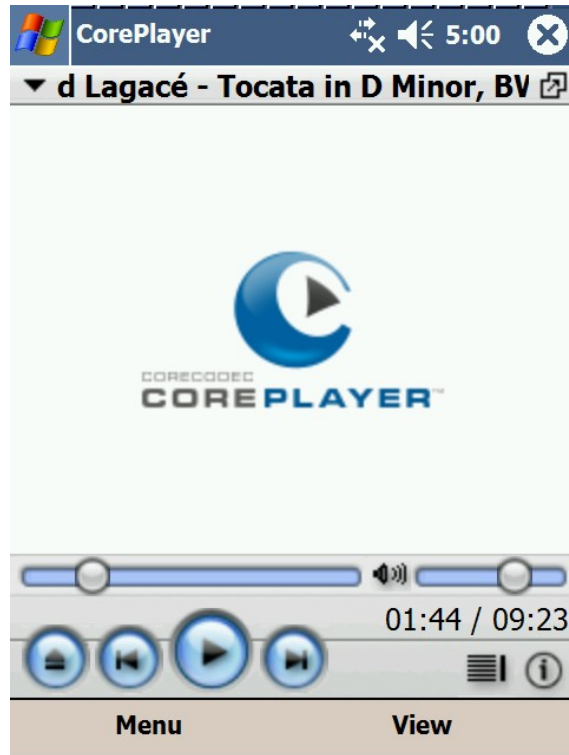


Figure 1: CP application window

Style

A STYLE element is basically a formatting element. When it is applied to other types of elements, it can set things like text size, color, and formatting (bold, underline, italic, etc), and background color. Most STYLE element declarations will come near the beginning of the skin declaration and will fall outside of the APP element structure.

Named elements:

Content / child elements:

None. A STYLE consists only of attributes.

Syntax:

- `<STYLE NAME [COLOR] [BACKCOLOR] [BOLD] [TEXTHEIGHT] />`

Attributes:

- **NAME:** Required. Data type -> string. Gives the STYLE a unique name that distinguishes it from other elements in the file.
- **COLOR:** Optional. Data type -> hexadecimal integer. Sets a color for the style.
- **BACKCOLOR:** Optional. Data type -> hexadecimal integer. Sets the background color.
- **BOLD:** Optional. Data type -> Boolean (1 or 0). Sets the boldness of the text (0=normal, 1=bold).
- **TEXTHEIGHT:** Optional. Data type -> integer. Sets the size of the text.

Example:

- `<STYLE NAME="title" COLOR="#000000" BOLD="1" TEXTHEIGHT="15" />`

This creates a STYLE named "title" with a black, bold, size 15 font.



Figure 2: the image on the left has a STYLE with bold font applied.

Mode

A MODE element defines a set of skin elements that is unique to a certain mode of operation in CorePlayer. An example of this would be the difference between the look of the player when it first opens and its look when playing a full-screen video. In each case, the elements of the skin are different (ie, in normal mode, there are Play / FF / REW controls, but no buttons at all in full-screen mode), and are each contained within a MODE element. Any number of modes can exist in a skin, depending on the preferences of the skin designer.

Named elements:

Content / child elements:

IMG, SLIDER, WINMOVE, BUTTON, LABEL, WINSIZE, FILL, VIEWPORT, and GROUP elements are all allowed as children in a MODE object.

Syntax:

- `<MODE NAME [DEFAULT]>`

Attributes:

- **NAME:** Required. Data type -> String. Gives the MODE a unique name that distinguishes it from other elements in the file.

- **DEFAULT:** Optional. Data type -> Boolean. Sets the MODE as the mode that the player will start in when opened. If none of the MODEs are set as the default, the player will start up in the first declared MODE.

Example:

- `<MODE NAME="playlist">`
`(content)`
`</MODE>`

This creates a MODE named “playlist.” This mode could conceivably be used to create the skin for when the player shows the current playlist while playing media files.

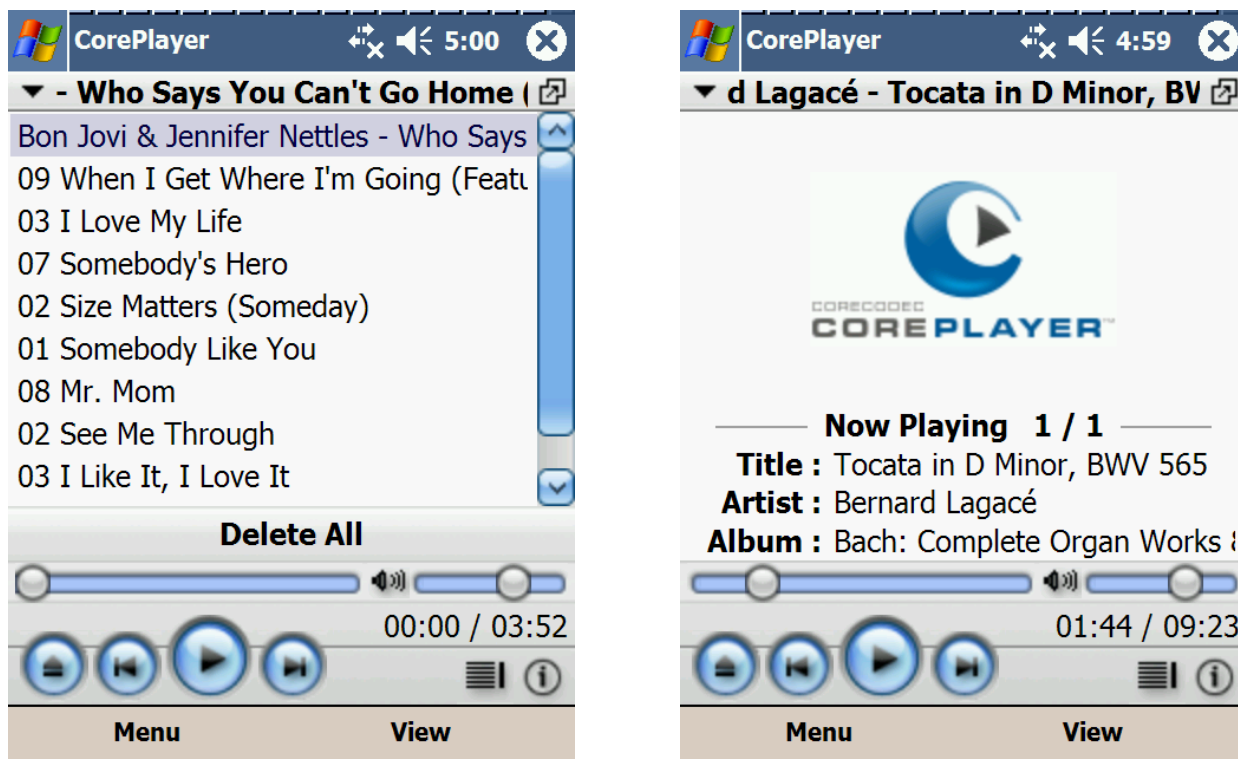


Figure 3: Two different MODEs within the CorePlayer APP

Group

A GROUP element takes all of its child elements and combines them together by applying a set of global position constraints (ie, four buttons positioned at the corners of a square could be grouped together and placed at the bottom right corner of the CP window). While each of the child elements can be positioned in relation to each other, their position in relation to objects higher in the element hierarchy is controlled by the GROUP attributes.

Named elements:

Content / child elements:

IMG, SLIDER, WINMOVE, BUTTON, LABEL, WINSIZE, FILL, and VIEWPORT elements are all allowed as children in a GROUP object.

Syntax:

- `<GROUP [SKIP] POSITION>`
 (content)
 `</group>`

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **POSITION:** Data type -> integer. Note that POSITION is itself not an attribute name, but encompasses a whole list of different attributes, most any combination of which can be used. These define the position of the group or object within its parent object, and include the following attributes (I will discuss them for all elements, not just GROUP):
 - **ALIGN:** Sets the alignment of the object and in some cases, will resize the object to fit the space. For example, if I have a background image I want to use behind the volume and play/FF/REW buttons, even if it is not wide enough to fit the whole screen width, if I set ALIGN="bottom", the XML parser will resize the image to extend the width of the screen. The resizing occurs for alignment values of "bottom", "top", and "center". **(verify this.)**
 - **HALIGN:** Sets the horizontal alignment of the object.
 - **VALIGN:** Sets the vertical alignment of the object.
 - **LEFT:** Sets the horizontal distance, in pixels, of the left side of the object from the top left corner of its parent object.
 - **RIGHT:** Sets the horizontal distance, in pixels, of the right side of the object from the top left corner of its parent object.
 - **TOP:** Sets the vertical distance, in pixels, of the top side of the object from the top left corner of its parent object.
 - **BOTTOM:** Sets the vertical distance, in pixels, of the bottom side of the object from the top left corner of its parent object.
 - **LEFTE:** Sets the horizontal distance, in pixels, of the left side of the object from the bottom right corner of its parent object. Note that this will in most cases be a negative number.
 - **RIGHT:** Sets the horizontal distance, in pixels, of the right side of the object from the bottom right corner of its parent object. Note that this will in most cases be a negative number.
 - **TOPE:** Sets the vertical distance, in pixels, of the top side of the object from the bottom right corner of its parent object. Note that this will in most cases be a negative number.

- **BOTTOMC**: Sets the vertical distance, in pixels, of the bottom side of the object from the center of its parent object. Note that this will in most cases be a negative number.
- **LEFTC**: Sets the horizontal distance, in pixels, of the left side of the object from the center of its parent object.
- **RIGHTC**: Sets the horizontal distance, in pixels, of the right side of the object from the center of its parent object.
- **TOPC**: Sets the vertical distance, in pixels, of the top side of the object from the center of its parent object.
- **BOTTOMC**: Sets the vertical distance, in pixels, of the bottom side of the object from the center of its parent object.

Example:

- `<GROUP SKIP="platform.desktop" ALIGN="BOTTOM" />`

When CorePlayer is installed on any device other than a desktop, a GROUP is created that is aligned at the bottom of the player window.



Figure 4: outlined in red is a GROUP that has been aligned at the bottom of the window

Image (IMG)

An IMG element is a container for an BITMAP that the parser can resize and/or only selectively display. It can be used to display background elements, buttons, sliders, and various other controls in CP.

Named elements:

Content / child elements:

None. An IMG element consists only of attributes.

Syntax:

- ``

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **VISIBLE:** Optional. Data type -> Boolean. Sets the condition under which the IMG is visible. One common use for this might be when the IMG is part of a button definition, as you can set one IMG to be visible when the button is in one state, and another IMG to be visible when it's in another state (for example, being clicked).
- **SRC:** Required. Data type -> string. Sets the source of the IMG to be one of the BITMAPS defined elsewhere in the skin definition.
- **POSITION:** Optional. Data type -> integer. See GROUP definition above.

Example:

- ``

This creates an image containing the “play” bitmap, and whose left side is 1 pixels from the right side of its parent control, and whose top is 1 pixels from the bottom of the parent control.



Figure 5: an IMG containing the “play” bitmap

Slider

A SLIDER element is used for displaying and/or setting CP variables that can have a whole range of values. For example, the volume indicator would be an ideal variable to map to a SLIDER because it can vary anywhere from zero volume to the current system volume.

Named elements:

Content / child elements:

IMG and THUMB elements are the allowable child elements of a SLIDER object.

Syntax:

- `<SLIDER [SKIP] NAME ACTION POSITION>`
 (content)
• `</SLIDER>`

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **NAME:** Data type -> string. Gives the SLIDER a unique name that distinguishes it from other elements in the file.
- **ACTION:** Data type -> CP global variable. Maps the slider to the value of a CorePlayer variable. Some common variables might include the following:
 - % complete (of file): "player.percent"
 - Volume level: "player.volume"
 - Playlist scroll: "settings.scrollv" or "list.scrollv"
- **POSITION:** Data type -> integer. See GROUP definition above.

Example:

- `<SLIDER ACTION="player.percent" LEFT="10" RIGHT="-170" TOP="3" />`

This creates a SLIDER that's mapped to the percent complete variable of the currently playing file. The left side of the SLIDER is 10 pixels to the right of the left side of its parent element, its right side is 170 pixels to the left of the right side of its parent element, and its top is 3 pixels below the top of its parent element.

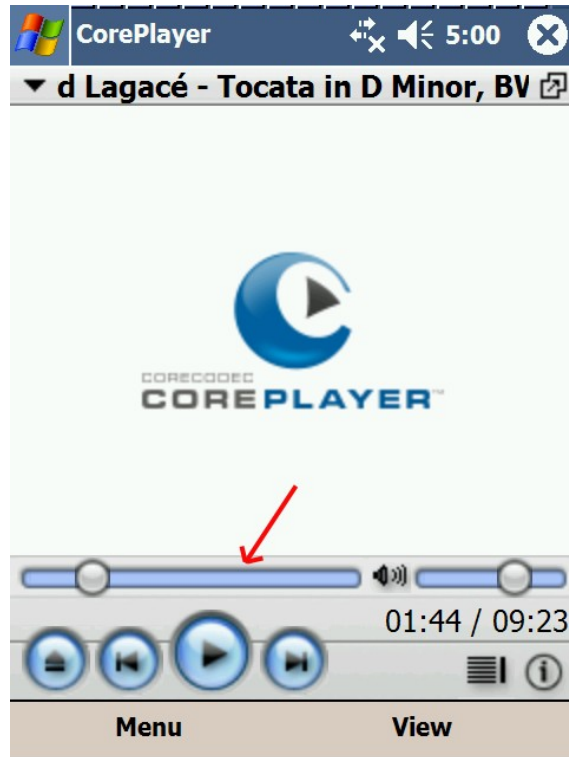


Figure 6: the red arrow is pointing to the SLIDER element

Thumb

A THUMB element is used exclusively with SLIDER elements and is the part of the slider that can be moved. It either displays the CP value that the SLIDER has been mapped to or sets that value when the user moves the THUMB control.

Named elements:

Content / child elements:

An IMG element is the only acceptable child element of a THUMB.

Syntax:

- `<THUMB>`
(IMG element)
`</THUMB>`

Attributes:

- No attributes.

Example:

- `<THUMB>`
 ``
 `</THUMB>`

This creates a THUMB that has the “slider_thumb” IMG mapped to it.

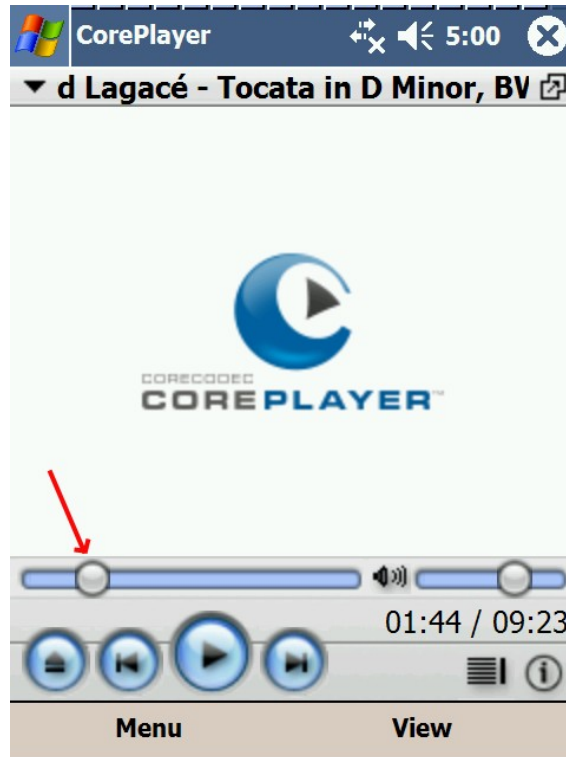


Figure 7: the red arrow is pointing to the THUMB element of the SLIDER

Window move control (WINMOVE)

A WINMOVE element is an object that when clicked on will allow the user to move the entire CP window (kind of like clicking on the top bar of a window on a desktop allows you to move the entire window if it's not maximized). It is not a very useful element for devices on which CP runs maximized (ie, most mobile devices).

Named elements:

Content / child elements:

None. A WINMOVE element consists only of attributes.

Syntax:

- `<WINMOVE [SKIP] POSITION />`

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **POSITION:** Data type -> integer. See GROUP definition above.

Example:

- `<WINMOVE SKIP="!platform.desktop" LEFT="160" RIGHT="-16" TOP="1" BOTTOM="-1" />`

This creates a WINMOVE control only on desktop devices. It's 160 pixels from the left, 16 pixels from the right, 1 pixel from the top and 1 pixel from the bottom of its parent element.

Button

A BUTTON element is a clickable control that performs a specific action when clicked. Pretty simple.

Named elements:

- pushed
- down

Content / child elements:

IMG, LABEL and FILL objects are acceptable child elements of a BUTTON. One common use of multiple IMG child elements would be to have different ones selectively visible based on whether the button is currently being clicked or not.

Syntax:

- `<BUTTON [SKIP] ACTION [VALUE] POSITION>`
 (content)
 `</BUTTON>`

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **ACTION:** Data type -> CP global function name (string). Sets the action that the button performs when clicked. Some common actions include the following:
 - "player.play"
 - "player.prev"
 - "player.next"

- "player.mute"
- "player.timerleft"
- "player.fullscreen"
- "filedialog.openmedia"
- "mode" (see VALUE attribute below)
- "minimized"
- "Close"
- "ok" (useful in dialogs)
- "cancel" (useful in dialogs)
- **VALUE:** Data type -> string. Required only if ACTION="mode". The value of this attribute must be the name of a declared MODE element in the skin file.
- **POSITION:** Data type -> integer. See GROUP definition above.

Example:

- ```
<BUTTON SKIP="platform.mouse" VALUE="player.play" LEFTC="0" TOPC="0" />

</BUTTON>
```

This creates a button that, when clicked, toggles its state between either playing or pausing the current media file. The "play\_up" IMG is visible when the button is in its "up" state, but when the user clicks the button, it toggles its state and the "play\_down" IMG is displayed instead.



**Figure 8: The play BUTTON in its up (left) and down (right) states**

## **Label**

A LABEL element creates a place where un-editable text can be placed. This is useful for things like displaying the currently playing title or the length of the song / movie.

Named elements:

Content / child elements:

None. A LABEL element consists only of attributes.

Syntax:

- `<LABEL [SKIP] TEXT STYLE POSITION [ANIMSCROLL] />`

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **TEXT:** Data type -> string. Sets the text of the label. This can be a hard-coded (ie, non-changing) piece of text or can be a global CP variable (preceded by a '\$'). Some common variables might be the following:
  - "\$player.timerandtotal"
  - "\$player.autotitle"
  - "\$player.listpos"
  - "\$player.title"
  - "\$player.artist"
  - "\$player.album"
- **STYLE:** Data type -> String. Formats the text of the LABEL element based on a STYLE element created elsewhere in the skin definition.
- **POSITION:** Data type -> integer. See GROUP definition above.
- **ANIMSCROLL:** Data type -> Boolean (1 or 0). Determines whether the label text scrolls if it's too long to fit (0=no, 1=yes).

Example:

- `<LABEL TEXT="$player.timerandtotal" STYLE="uitext" HALIGN="right" />`

This creates a label that's horizontally aligned to the right side of its parent element. Its text shows the total length of the currently playing file, and is formatted according to the STYLE element "uitext".

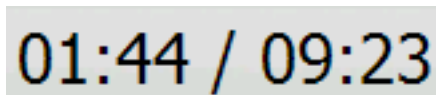


Figure 9: a LABEL that displays elapsed time vs. total time

### Window resize control (WINSIZE)

A WINSIZE control allows the user to resize a non-maximized window by clicking and dragging it. It isn't very useful on devices where CP runs maximized (most PDA's). This element should be placed at the bottom right corner of the window to make the CP window consistent with other resizable application windows.

Named elements:

Content / child elements:

IMG, LABEL, and FILL objects are all acceptable child elements of a WINSIZE object. However, of the three, an IMG is probably the best way to visualize the control.

Syntax:

- **<WINSIZE** [SKIP] POSITION>  
    (*content*)  
  </WINSIZE>

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **POSITION:** Data type -> integer. See GROUP definition above.

Example:

- **<WINSIZE LEFT=-10" TOPE=-10">**

This creates a WINSIZE control whose left side is 10 pixels from the right side of its parent control, and whose top is 10 pixels from the bottom of the parent control.

## **Fill**

A FILL element is a visual element that sets the background color of its parent element.

Named elements:

Content / child elements:

None. A FILL element consists only of attributes.

Syntax:

- **<FILL** [SKIP] STYLE [POSITION] />

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **STYLE:** Data type -> string. Sets the STYLE of the FILL element. The STYLE element chosen should ideally be one that just sets a background color.
- **POSITION:** Optional. Data type -> integer. See GROUP definition above.

Example:

- **<FILL STYLE="back"/>**

This creates a FILL element with the STYLE named “back”.

## Viewport

A VIEWPORT is the element where the video in a file is shown, and can be conditionally visible based on whether the current file is a video or not.

Named elements:

Content / child elements:

None. A VIEWPORT consists of only attributes.

Syntax:

- `<VIEWPORT [VISIBLE] STYLE [MOUSEHIDE] />`

Attributes:

- **VISIBLE:** Optional. Data type -> Boolean. Sets the conditions under which the VIEWPORT is visible.
- **STYLE:** Data type -> string. Sets the STYLE of the VIEWPORT element.
- **MOUSEHIDE:** Optional. Data type -> integer. Sets the number of ticks (or seconds) that a mouse cursor is hidden when it passes over the VIEWPORT.

Example:

- `<VIEWPORT VISIBLE="player.isviewport" STYLE="viewport"/>`

This creates a VIEWPORT that is visible only if CorePlayer is currently playing a video file (player.isviewport).



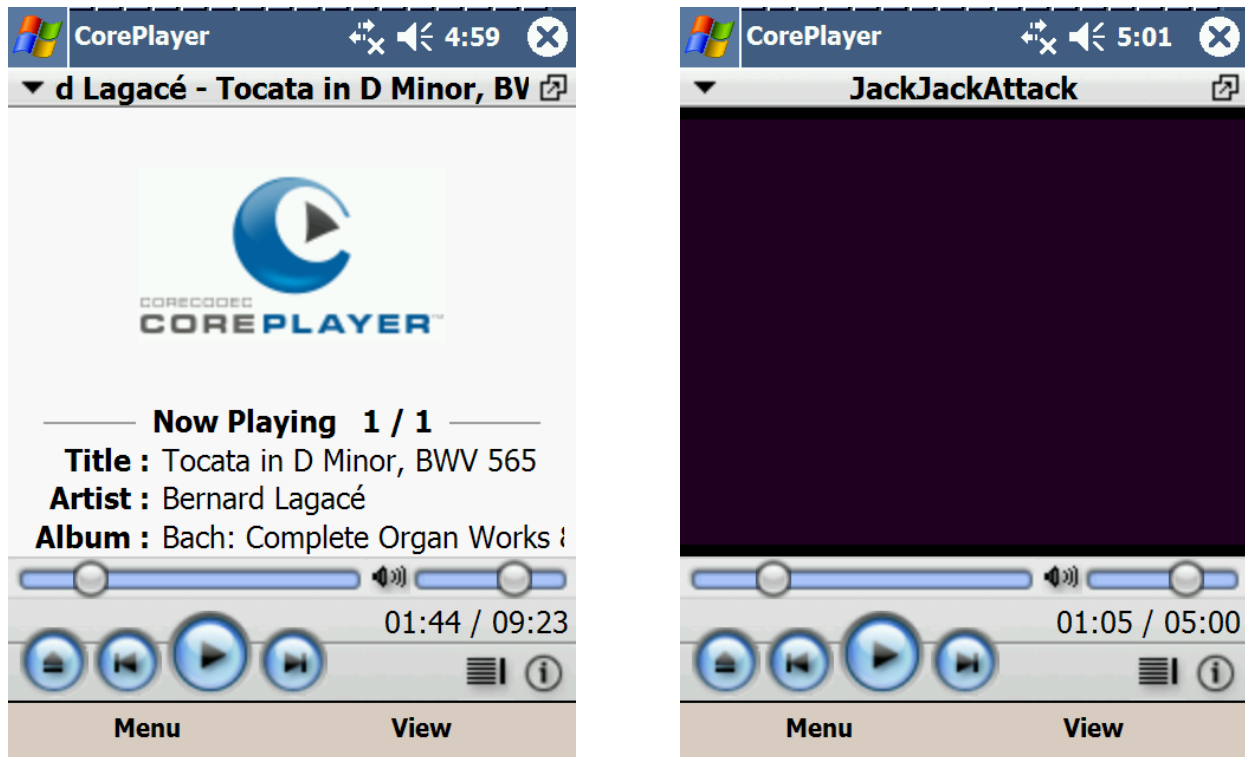


Figure 10: here we see that the VIEWPORT is only visible when a video is playing (right).

## Enumbox

An enumbox is an element used to display the contents of an enumerated list. An enumerated list is just an array or list that has named elements in a specific order so that they can be referenced numerically. In CP, an ENUMBOX looks like a scrollbar where values cycle through it.

Named elements:

None.

Content / child elements:

STYLE, GROUP, IMG, FILL, POPUPMENU and BUTTON are all acceptable child elements of an ENUMBOX control.

Syntax:

```
<ENUMBOX [TABSTOP] [POPUP] />
```

Attributes:

- **TABSTOP:** Optional. Data type -> boolean. Determines whether the user using the tab key can give focus to this control or not (1=yes).
- **POPUP:** Optional. Data type -> string. (**what does this do?**)

Example:

- `<ENUMBOX TABSTOP="1" POPUP="pop">`  
`(content)`  
`</ENUMBOX>`

This example creates an ENUMBOX that can receive focus from the tab key, and which has the POPUP attribute set to "pop".

## Popupmenu

A popupmenu is a control is a button which will display a menu when clicked. This type of control allows for including menus in a skin in places other than the bottom portion of a mobile device, and also allows the menus themselves to be skinned, as buttons can be designed to look however the skin designer wants.

Named elements:

None.

Content / child elements:

Any available class type except DIALOG, MENU, APP and HOTKEY.

Syntax:

`<POPUPMENU NAME ENUM [ALIGN] [POSITION] [ATTACHED] >`

Attributes:

- **NAME:** Required. Data type -> string. Gives the POPUPMENU a unique name that distinguishes it from other elements in the file.
- **ENUM:** Required. Data type -> string. Sets the name of the ENUM that should be used to populate the POPUPMENU
- **ALIGN:** Optional. Data type -> string. Sets the alignment of the POPUPMENU.
- **POSITION:** Optional. Data type -> integer. See GROUP definition above.
- **ATTACHED:** Optional. Data type -> string. Links the POPUPMENU to another control. (**what does this do?**)

Example:

- `<POPUPMENU NAME="pop" ENUM="enumvalue" ALIGN="center" ATTACHED="parent">`  
`(content)`  
`</POPUPMENU>`

This example creates a POPUPMENU element named “pop”, populated with the values from the ENUM “enumvalue”, aligned at the top of its parent element and attached to it.

## Split

A split element creates a control that splits two other groups of controls either horizontally or vertically. The user can change the location of the split line by dragging it, which changes the respective sizes of the two groups.

Named elements:

None.

Content / child elements:

Any available class type except DIALOG, MENU, APP and HOTKEY.

Syntax:

`<SPLIT [HORIZONTAL] [VERTICAL] [PROPORTIONAL] [ALIGN] [POSITION] />`

Attributes:

- **HORIZONTAL**: Optional. Data type -> Boolean. Sets the SPLIT direction to be horizontal.
- **VERTICAL**: Optional. Data type -> Boolean. Sets the SPLIT direction to be vertical.
- **PROPORTIONAL**: Optional. Data type -> Boolean. Determines whether the shape of the SPLIT control remains constant upon resizing.
- **ALIGN**: Optional. Data type -> string. Sets the alignment of the SPLIT.
- **POSITION**: Optional. Data type -> integer. See GROUP definition above.

Example:

- `<SPLIT VERTICAL="1" ALIGN="left">`  
`(content)`  
`</SPLIT>`

This creates a vertically oriented SPLIT aligned on the left hand side of its parent control.

## *Splitbar*

A splitbar is the actual line that splits the two sides of a SPLIT control. It is useful because it allows the designer to change the thickness of the bar.

Named elements:

None.

Content / child elements:

None. This element has attributes only.

Syntax:

```
<SPLITBAR [WIDTH] [HEIGHT] />
```

Attributes:

- **WIDTH:** Optional. Data type -> integer. Sets the width of the splitbar in pixels.
- **HEIGHT:** Optional. Data type -> integer. Sets the height of the splitbar in pixels.

Example:

- `<SPLITBAR WIDTH="4" />`

This creates a splitbar 4 pixels thick within its parent SPLIT element.

## *Splititem*

A splititem contains one of the two groups of controls that make up a SPLIT control (so each SPLIT can have at most two SPLITITEM controls in its content section).

Named elements:

None.

Content / child elements:

Any element type that's allowable within a GROUP control.

Syntax:

```
<SPLITITEM [WIDTH] [HEIGHT] />
```

Attributes:

- **WIDTH:** Optional. Data type -> integer. Sets the width of the splititem in pixels.
- **HEIGHT:** Optional. Data type -> integer. Sets the height of the splititem in pixels.

Example:

- `<SPLITITEM HEIGHT="80">`  
    *(content)*  
    `</SPLITITEM>`

This creates a SPLITITEM with a height of 80 pixels within its parent SPLIT.

## Tabkey

A tabkey is a place holder for a control that allows it to handle the tab key when it's being used to cycle through the controls in the skin.

Named elements:

None.

Content / child elements:

Any element type that's allowable within a GROUP control.

Syntax:

`<TABKEY>`

Attributes:

- **none**

Example:

- `<TABKEY>`  
    *(content)*  
    `</TABKEY>`

This creates a tab-key handler for its parent control.

## Properties

A properties element creates a special properties control for either the equalizer settings or the color settings.

Named elements:

None.

Content / child elements:

???

Syntax:

**<PROPERTIES** NAME SRC />

Attributes:

- **NAME:** Required. Data type -> string. Gives the PROPERTIES a unique name that distinguishes it from other elements in the file.
- **SRC:** Required. Data type -> string. Must be either “equalizer” or “color”.

Example:

- **<PROPERTIES** NAME="plist" SRC="color" />

This creates a PROPERTIES control named “plist” and mapped to the color settings.

## Poly

A poly control is a control that allows a designer to create a graphical control without using an IMG control. In the declaration of the POLY element is a list of points that get connected together with lines to create the polygon.

Named elements:

None.

Content / child elements:

None. This element is composed of attributes only.

Syntax:

**<POLY** POINTS />

Attributes:

- **POINTS:** Required. Data type -> string. The string contains a list of points in the form “(x1,y1),(x2,y2),(x3,y3),...”

Example:

- `<POLY POINTS="(2,2),(10,2),(10,10),(9,10),(9,3),(2,3)" />`

This creates a POLY with the six points connected together to form a polygon on the parent control.

## Dialog

A DIALOG element sets up the skin interface for one of four types of built-in dialog forms in CorePlayer. These built-in dialogs are the following:

- URL dialog – it’s here that a user can choose a website to stream media from.
- Settings dialog
- About dialog
- Serial dialog – here’s where the user enters his/her registration information.

Named elements:

Content / child elements:

IMG, SLIDER, BUTTON, LABEL, WINSIZE, and FILL objects are acceptable child elements in a DIALOG.

Syntax:

- `<DIALOG [SKIP] NAME WIDTH HEIGHT [MINWIDTH] [MINHEIGHT] SOFTMENU [CAPTION]>`  
    *(content)*  
    `</DIALOG>`

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **NAME:** Data type -> string. Gives the DIALOG a unique name that distinguishes it from other elements in the file.
- **WIDTH:** Data type -> integer. Sets the width of the DIALOG in pixels.
- **HEIGHT:** Data type -> integer. Sets the height of the DIALOG in pixels.
- **MINWIDTH:** Optional. Data type -> integer. Sets the smallest allowed width of the DIALOG in pixels.
- **MINHEIGHT:** Optional. Data type -> integer. Sets the smallest allowed height of the DIALOG in pixels.

- **SOFTMENU:** Data type -> string. Sets the MENU to use with the DIALOG (the MENU must have been declared elsewhere in the skin file).
- **CAPTION:** Optional. Data type -> boolean. Sets whether the caption of the DIALOG is visible (1=yes, 0=no).

Example:

- `<DIALOG NAME="aboutbox" WIDTH="320" HEIGHT="110" SOFTMENU="aboutbox_softmenu" CAPTION="0">`  
`(content)`  
`</DIALOG>`

This creates an about-box dialog with dimensions of 100x320, with no caption, and with the “aboutbox\_softmenu” menu.

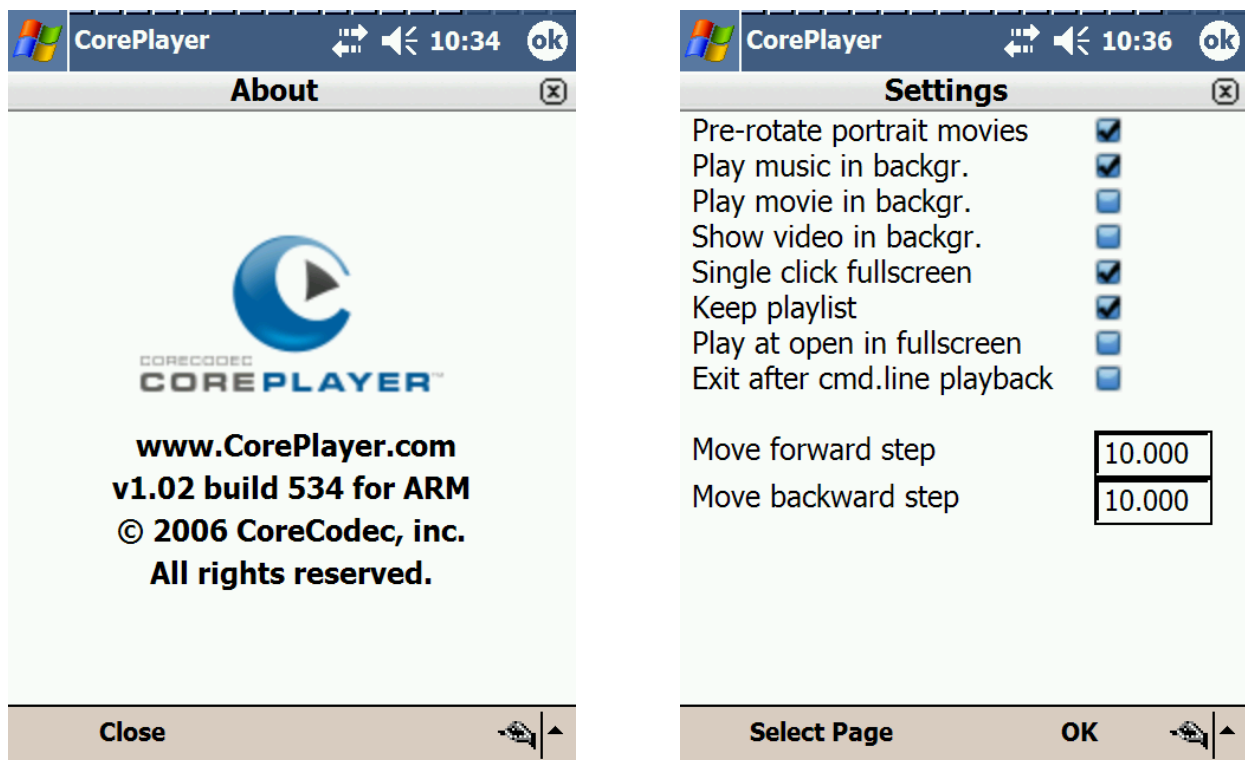


Figure 11: about box DIALOG & settings DIALOG of CorePlayer

## Menu

A MENU element creates an entire nested menu structure, and so can be quite complicated. An arbitrary number of MENU structures can be created for different dialog forms.

Named elements:



Content / child elements:

Other MENU objects are the only acceptable child elements of a MENU structure.

Syntax:

- `<MENU [SKIP] NAME BASE />`  
OR  
`<MENU [SKIP] TEXT [ACTION] />`

Note: the first statement should only be used when declaring the entire menu structure. The second statement is used for creating menu items.

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **NAME:** Data type -> string. Gives the MENU structure a unique name that distinguishes it from other elements in the file.
- **BASE:** Data type -> string. Sets the name of the base APP or DIALOG that the MENU structure will be used in.
- **TEXT:** Data type -> string. Either hard-codes the text of the menu item or goes to a language file and looks up a value string (in this case, the value must be preceded by a '#' symbol).
- **ACTION:** Optional only if the MENU item contains sub menus. Data type -> CP function name (string). Sets the action that the MENU performs when clicked.

Example:

- ```
<MENU NAME="main_softmenu" BASE="main">
  <MENU TEXT="#MENU0100">
    <MENU TEXT="#MENU0101" ACTION="filedialog.openmedia"/>
  </MENU>
</MENU>
```

This example creates a MENU structure called “main_softmenu” that will be used with the APP or DIALOG named “main”. It has a single item, which in turn has a single sub-item. This sub-item opens a file dialog box to select media to play.

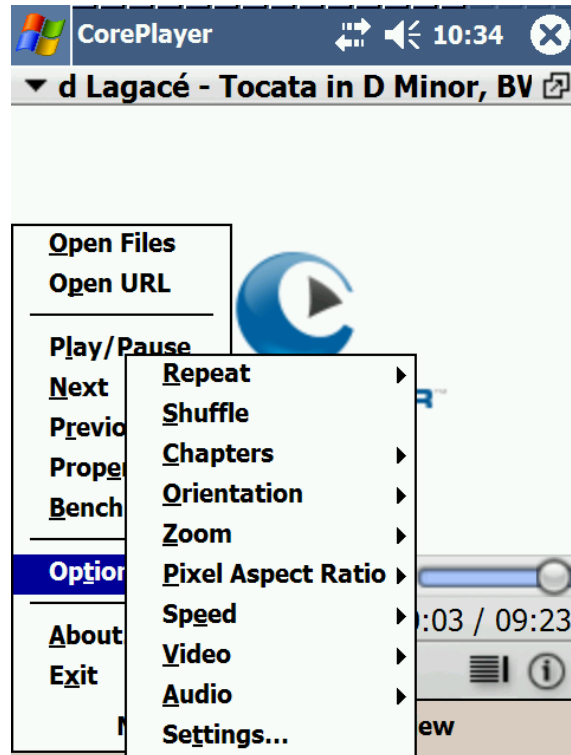


Figure 12: “main_softmenu” MENU

Shortcut (hotkey)

A HOTKEY element defines a list of keyboard combinations that act as shortcuts to different CorePlayer functions.

Named elements:

Content / child elements:

Only other HOTKEY elements may be used with a HOTKEY structure.

Syntax:

- `<HOTKEY [SKIP] NAME BASE />`

OR

`<HOTKEY [SKIP] KEY ACTION [VALUE] [DELTA] [CYCLE] />`

Note: the first statement should only be used when declaring the entire hotkey structure. The second statement is used for creating specific hotkey combinations.

Attributes:

- **SKIP:** Optional. Data type -> Boolean. Sets the condition where this element statement is skipped over by the parsing engine (see BITMAP definition above).
- **NAME:** Data type -> string. Gives the HOTKEY structure a unique name that distinguishes it from other elements in the file.
- **BASE:** Data type -> string. Sets the name of the base APP or DIALOG that the MENU structure will be used in.
- **KEY:** Data type -> string. Sets the key combination that will invoke the specified function.
- **ACTION:** Data type -> CP function name (string). The action that the HOTKEY performs.
 - (*include here the types of actions that can be performed?*)
- **VALUE:** Optional. Data type -> string. Depending on the type of ACTION, a value may need to be set (example, setting ACTION="player.speed" requires VALUE="10:100" or "100:100", etc.)
- **DELTA:** Optional. Data type -> integer. Depending on the type of ACTION (for example, increasing or decreasing volume), a DELTA needs to be set.
- **CYCLE:** Optional. Data type -> integer. Used if VALUE="player.repeat"

Example:

- ```
<HOTKEY NAME="main_hotkey" BASE="main" >
 <HOTKEY KEY="escape" ACTION="player.fullscreen" VALUE="0" />
</HOTKEY>
```

This creates a HOTKEY structure with name "main\_hotkey" and linked to the APP or DIALOG with the name of "main". It declares a hotkey combination that ESC will exit fullscreen mode.

## Skin structure

Ok, now we know what possible element types are available to us for building a custom CorePlayer skin, together with a simple example of how to use each one of them. However, how do all of these fit together in the macro-structure of the skin definition? There isn't necessarily a hard-and-fast way to do it, but below I outline a structure that will keep the file simple and well-organized.

- Define bitmap names (BITMAP)
- Define application window properties, menu, etc. (APP)
  - Define text, background & other styles (STYLE)
  - Define interface for different modes (video, fullscreen, meta, playlist) (MODE)
    - Define groups of controls/elements
      - Add elements to groups
        - Possible elements include:

- Image (IMG)
  - Slider (SLIDER)
    - Thumb to indicate slider position (THUMB)
  - Window movement (WINMOVE)
  - Button (BUTTON)
  - Label (LABEL)
  - Window resize control (WINSIZE)
  - Fill element (FILL)
  - Viewport to display video (VIEWPORT)
- (end application)
- Define interface for different dialogs (URL, settings, about, files, benchmark, mediainfo, etc.)
  - (Define here which elements various dialogs require to be present. For example, the main window requires a play/pause, FF, REW keys to be present)
- Define menus for each of the dialogs (including main window)
- Define hotkeys for each of the various platforms

Again, there is no hard and fast element structure, since the structure will change based on your skinning preferences and design decisions. However, you can use the above as a general guideline if you're just starting to skin, or, if you'd rather be adventurous and do something completely different, feel free to do so!

## File structure and names

Finally, once your skin definition and your bitmap files have been prepared, all you have to do to finish and install the skin is put the files into their proper places. The name of the XML definition must be "skin.xml" and this file, along with all of its dependant bitmap images, may be compressed into a zip file names "skin.zip" or "skin.CPS" if you choose to do so. If not, you may leave them all in their non-compressed format, but they will obviously require more memory on the device that CorePlayer is installed on, and the end user will have to deal with multiple files instead of just one during installation. Thus, we recommend that you compress all of the files into a single zip file.

To install the skin, either the single zip file or all of the compressed files must be stored in the same directory that "CorePlayer.exe" is located (the installation directory the user specifies when installing). CorePlayer, when it starts, will look for that file, and if it finds it, will parse it and use that skin. If no skin has been installed, CP will use the default internally defined skin that it ships with. And that's all!

## Conclusion

What we have just covered in this skinning guide are all of the basics that you need to start designing Universal Skins for CorePlayer. If you have questions about how to use the guide, or other skinning-related questions in general, feel free to post them at the skinning forums at CoreCodec (located at <http://www.corecodec.com/forum/index.php?board=26.0>). Thanks for your interest in creating CP Universal Skins – now, let your imagination run wild and see what comes of it!



## Appendix A: CorePlayer variable reference

List of possible items that can be referenced directly (for ex player.play):

- player (PLAYER\_ID)
- platform (PLATFORM\_ID)
- advanced (ADVANCED\_ID)
- color (COLOR\_ID)
- equalizer (EQUALIZER\_ID)
- network (NETWORK\_ID)
- buffers (PLAYER\_BUFFER\_ID)
- audio (PLAYER\_AUDIO\_ID)
- video (PLAYER\_VIDEO\_ID)

List of properties that can be referenced for these classes :

- PLAYER\_ID :
  - \* AutoPreRot (bool)
  - \* Repeat (enum: off(0),all(1),track(2), notify)
  - \* Shuffle (bool, notify)
  - \* KeepPlyAud (bool)
  - \* KeepPlyVid (bool)
  - \* ShowInBack (bool)
  - \* OneClickFull (bool)
  - \* KeepList (bool)
  - \* PlayAtOpen (bool)
  - \* PlayAtOpenFull (bool)
  - \* ExitAtEnd (bool) {windows only}
  - \* FullZoom (fraction)
  - \* SkinZoom (fraction)
  - \* Zoom (fraction)
  - \* SecondRun (bool)
  - \* Benchmark (bool)
  - \* BenchTime (tick)
  - \* Speed (fraction)
  - \* FFSpeed (fraction)
  - \* FwdStep (tick)
  - \* BwdStep (tick)

- \* ListCount (int)
- \* ListCurr (int)
- \* ListIdx (int)
- \* Aspect (fraction)
- \* FullDir (int)
- \* SkinDir (int)
- \* AStrm (enum)
- \* VStrm (enum)
- \* SStrm (enum)
- \* Chapter (enum)
- \* Percent (fraction, notify)
- \* TimerLeft (bool)
- \* FindFiles (event)
- \* Dump (event)
- \* Background (bool)
- \* Foreground (bool)
- \* Play (bool, notify)
- \* FFwd (bool, notify)
- \* Position (tick)
- \* Duration (tick)
- \* Timer (string, notify)
- \* TimerAndTotal (string, notify)
- \* ListPos (string, notify)
- \* AutoTitle (string, notify)
- \* Title (string, notify)
- \* Artist (string, notify)
- \* Genre (string, notify)
- \* Album (string, notify)
- \* Year (string, notify)
- \* Author (string, notify)
- \* Comment (string, notify)
- \* Input (node)
- \* Format (node)
- \* AOutput (node)
- \* VOutput (node)
- \* SkinViewport (rect)
- \* Clipping (bool)



- \* Fullscreen\_Play (bool)
- \* Fullscreen (bool, notify)
- \* LoadMode (bool, readonly)
- \* CurrentDir (string)
- \* Next (event)
- \* Prev (event)
- \* NextFile (event)
- \* PrevFile (event)
- \* Stop (event)
- \* CloseMedia (event)
- \* URL (string)
- \* Forward (event)
- \* Backward (event)
- \* IsViewport (bool, notify)
- \* Invalidate (event, notify)
- \* ProcessLoop (event, notify)
- \* Open (string)
- \* OpenMore (array)

#### - PLAYER\_BUFFER\_ID

- \* BufSize (int)
- \* VUnderrun (int)
- \* AUnderrun (int)
- \* MDMode (bool) {windows only}
- \* MDSize (int) {windows only}
- \* MDStart (int) {windows only}

#### - PLAYER\_AUDIO\_ID

- \* Mute (bool, notify)
- \* Volume (bool, notify)
- \* Pan (int) {Palm OS only}
- \* PreAmp (int)
- \* AQual (int,enum)
- \* Stereo (int,enum)
- \* AOut (fourCC, enum)
- \* AOutMax (fourCC)

- PLAYER\_VIDEO\_ID

- \* VOut (fourCC, enum)
- \* VQual (int, enum)
- \* Smooth (int, enum)
- \* Dither (bool, notify)
- \* VOutAccel (bool, notify)
- \* VOutMax (fourCC)

- PLATFORM\_ID

- \* ProjName (string, read only)
- \* ProjVendor (string, read only)
- \* ProjVersion (string, read only)
- \* HelpFile (string, read only)
- \* ProjFourCC (fourCC, read only)
- \* BuildVersion (int, read only)
- \* Lang (fourCC, enum)
- \* TypeName (string, read only)
- \* Ver (int, read only)
- \* VerName (string, read only)
- \* OemInfo (string, read only)
- \* Type (int, read only)
- \* Model (int, read only)
- \* CPUCaps (int, read only)
- \* CPU (string, read only)
- \* Mouse (bool, read only)
- \* Desktop (bool, read only)
- \* HiddenMenu (bool, read only)
- \* BuildText (string, read only)
- \* Help (event)
- \* CPUMhz (int, read only)
- \* WMV (int, read only)
- \* Card (bool) {Palm OS only}

- ADVANCED\_ID

- \* NoLight (bool)
- \* HomeScr (bool)
- \* OldShell (bool) {win CE only}

- \* SlowVid (bool) {x86 only}
- \* IDCTSwap (bool) {x86 & ARM only}
- \* Lookup (bool) {x86 & ARM only}
- \* KeyDir (bool)
- \* Pri (bool) {multithread only}
- \* SysVol (bool) {arm or ARM and not Palm only}
- \* VR41XX (bool) {MIPS only}
- \* BenchPos (bool)
- \* AviFps (bool)
- \* WidAud (bool) {win CE only}
- \* WidDLL (bool) {win CE only}
- \* AllKeys (bool) {win CE only}
- \* NoBattWarn (bool) {Palm only}
- \* NoEventChk (bool) {Palm only}
- \* CardPlugins (bool) {Palm only}
- \* BlinkLED (bool) {Palm only}
- \* MemHack (bool) {Palm only}
- \* NoDeblk (bool)
- \* AVOfs (tick)
- \* DropTol (tick)
- \* SkipTol (tick)

#### - COLOR\_ID

- \* Bri (int, notify)
- \* Con (int, notify)
- \* Sat (int, notify)
- \* R (int, notify)
- \* G (int, notify)
- \* B (int, notify)
- \* Caps (int)

#### - EQUALIZER\_ID

- \* Use (bool, notify)
- \* VolNorm (bool, notify)
- \* PreAmp (int)
- \* f1 (int, notify)
- \* f2 (int, notify)

- \* f3 (int, notify)
- \* f4 (int, notify)
- \* f5 (int, notify)
- \* f6 (int, notify)
- \* f7 (int, notify)
- \* f8 (int, notify)
- \* f9 (int, notify)
- \* f10 (int, notify)

- NETWORK\_ID

- \* Type (int, enum)
- \* Proxy (int, enum)
- \* ProxyAddr (string)
- \* ProxyPort (string)

## Appendix B: CorePlayer language file

;CHARSET=CP1252

[EN]

TUNY0000=Tuny Engine

DUMA0000=Dump

DUMV0000=Dump

TSFM0000=TS streams (TS)

TSST0000=TS streams (TS)

GE2D0000=Sony GE2D

S1D10000=EPSON Graphics Controller

ATI40000=ATI Overlay

ATII0000=ATI Decoder

OCON0000=Ascii Console

AMRN0000=GSM AMR-NB

AMRW0000=GSM AMR-WB

AMFN0000=GSM AMR-NB files (AMR)

AMFW0000=GSM AMR-WB files (AWB)

ASAP0000=Another Slight Atari Player (SAP,CMC,CMR,DMC,MPT,MPD,RMT,TMC,TM8,TM2)

MIKM0000=Module files (XM,S3M,MOD,STM,IT,ULT,IMF,AMF,669)

MPCF0000=Musepack files (MPC)

MP4F0000=MPEG4 files (MP4,MOV,3GP,M4A,M4B,K3G,MQV)

MP4F0200=This type of MPEG4 file is not supported.

MP4F0201=Multiple media formats per one stream is not supported.

AAC\_0000=AAC files (AAC)

FAAD0000=LibFAAD LC,HE,PS AAC

FAAD0200=This type of AAC audio stream is not supported.

CAAC0200=This type of AAC audio stream is not supported.

A52\_0000=LibA52 (AC-3)

AC3\_0000=AC-3 Audio files (AC3)

TIFF0000=TIFF

TIFF0200=This type of TIFF image format is not supported.

RTIF0000=TIFF files (TIF,TIFF)

CVID0000=Cinepak

CRAM0000=Microsoft Video 1

PNG\_0000=PNG  
PNG\_0200=This type of PNG image format is not supported.  
RPNG0000=PNG files (PNG)  
RJPG0000=JPEG files (JPG,JPEG)  
WJPG0000=WEB MJPEG stream  
MJPG0000=MJPEG  
MJPG0200=This type of motion-JPEG format is not supported.  
MJPG0201=Progressive JPEG format is not supported yet.  
RASX0000=Windows metafiles (ASX,WMX,WVX,WAX)  
M3U\_0000=M3U playlist files (M3U)  
PLS\_0000=PLS playlist files (PLS)  
B4S\_0000=B4S playlist files (B4S)  
MATR0000=Matroska files (MKV,MKA)  
RSKN0000=Skin files (XML)  
RNSV0000=NSV files (NSV)  
RYUV0000=YUV files (YUV)  
RAVI0000=Video files (AVI,DIVX,GVI)  
RAVI0100=Index corrupt. No seeking possible!  
RWAV0000=Windows waveform (WAV,RMP)  
RAIF0000=Audio Interchange File Format (AIF)  
RASX0000=ASF media files (ASF)  
RASH0000=ASF HTTP stream  
WMAF0000=Windows Media Audio files (WMA)  
WMVF0000=Windows Media Video files (WM,WMV)  
WMV\_0000=DMO WMV  
WMVA0000=DMO WMV Advanced profile  
WMA\_0000=DMO WMA  
WMAV0000=DMO WMA Voice  
WMVD0000=Windows Media Video  
WMAD0000=Windows Media Audio  
AVC\_0202=This type of AVC file is not supported.  
AVC\_0203=Interlaced video is not supported!  
AVC\_0204=FMO feature is not supported!  
AVC\_0206=Maximum %d pixel width video supported!  
AVC\_0207=MBAFF interlaced video is not supported!  
AVC\_0208=High profile is not supported!  
RAVC0000=AVC elementary stream (H264)

DIVX0200=Quarter Pixel feature is not supported!  
DIVX0201=Interlaced video is not supported!  
DIVX0202=GMC feature is not supported!  
DIVX0204=This type of H.263 video is not supported!  
DIVX0205=Data partitioning feature is not supported!  
VOUT0100=Maximum %d x %d video size is supported!  
MPEG0000=MPEG Movie files (MPEG,MPG,MPV)  
MPEG0200=Encrypted MPEG-2 streams are not supported!  
MVES0000=MPEG Video elementary stream (M1V,M4V)  
MPG10000=MPEG Video  
MPG10201=MPEG-2 video is not supported!  
MPG10202=Interlaced MPEG-2 video not supported!  
MPG10203=Chorma format not supported!  
MP3\_0000=MPEG Audio files (MP1,MP2,MP3,MPA)  
LMAD0000=LibMad MPEG Audio  
VORV0000=Vorbis Audio  
OGGE0000=Vorbis Audio (Embedded)  
OGGP0000=Vorbis Audio (Packed)  
OGGV0000=Ogg Vorbis files (OGG,OGM)  
SPXL0000=Speex files (SPX)  
SPEX0000=Speex speech decoder  
AHII0000=ATI IMAGEON Decoder  
AHI\_0000=ATI IMAGEON  
AHI\_0111=Disable OS Bitmap Caching (Advised!)  
AHI\_0110="Green Tint" bug compensation  
AHI\_0113=Reuse primary surface if needed  
AHI\_0114=Optimized primary surface mode  
AHI\_0115=Scaling LCD tearing compensation  
AHI\_0116=Keep ATI driver active (just test)  
AHI\_0117=ATI older driver memory mapping fix  
AHI\_0200=Out of video memory! Retry after soft-reset or choose a different video driver!  
AHI\_0201=Device open error! Retry after a warm boot (soft-reset)!  
AHI\_0202=Too old ATI driver (%s). Please update! In most cases the SDIO interface update will contain the new ATI driver. Until then try to using the player with GAPI option  
270G0000=Intel 2700G  
270G0400=Intel 2700G chip is not available, most likely used by another program!  
270I0000=Intel 2700G Decoder

270G0200=LCD tearing compensation  
270G0201=Decoder rotation (faster)  
270G0202=Close WMP when resource is locked  
GAPI0200=GAPI not supported on this device. Try a different driver!  
OSYM0000=Display  
HIRS0000=Display  
HIRS0100=Triple buffering. Needs more memory. Only visible on 320x480 screens.  
HIRS0101=LCD white borderless fullscreen. Not all LCD panels support it. UNCHECK IF IT DOESN'T WORK!  
DRAW0100=Overlay with colorkey  
DRAW0101=Use blitting instead of overlay  
DRAW0102=Use device stretching for blitting  
DRAW0103=Overlay format  
DRAF0000=Auto  
DRAF0001=YV12  
DRAF0002=YUY2  
DRAF0003=RGB  
XSC20000=Intel XScale  
XSL20000=Intel XScale Low Quality  
XSCD0000=Intel XScale  
XSCD0100=Hardware vertical zooming  
XSCD0101=Buffer flipping (solves LCD tearing)  
XSCD0102=Allocate flipping buffers at startup  
RAWD0000=Raw FrameBuffer  
HTTP0000=HTTP protocol  
MMSH0000=MMS protocol  
MMS\_0000=MMS protocol  
TCP\_0000=TCP protocol  
UDP\_0000=UDP protocol  
FILE0000=File  
FIDB0000=FileDb  
VFS\_0000=VFS  
STRM0091=URL  
WAVE0000=Wave Output  
NULV0000=Null Video  
NULA0000=Null Audio  
FMT\_0033=Duration



FMT\_0044=Filesize  
PLAY0000=Player  
PLAR0000=Off  
PLAR0001=All  
PLAR0002=Track  
PLAQ0000=Low  
PLAQ0001=Medium  
PLAQ0002=High  
PLAO0000=Off  
PLAO0001=Only for 50%  
PLAO0002=On  
PLAS0000=Stereo  
PLAS0001=Stereo swapped  
PLAS0002=Mono Join  
PLAS0003=Mono Left  
PLAS0004=Mono Right  
PLAY0108=Audio codec (%s) not supported by the player!  
PLAY0109=Video codec (%s) not supported by the player!  
PLAY010B=%s decoder not included! It was removed from the official install package because of intellectual property considerations.  
PLAY010C=%s decoder not included! It was removed from the official install package because of intellectual property considerations.  
PLAY0106=%s\nUnknown file format!  
PLAY0107=Buffer size is too small for this media. Try increase it in player settings dialog for better playback!  
PLAY010A=Buffer underun occured. The storage media is too slow, the player will periodical pause for loading. Try increase buffer size in settings for better playback!  
PLAY0110=Title  
PLAY0111=Artist  
PLAY0112=Album  
PLAY0113=Language  
PLAY0114=Genre  
PLAY0115=Author  
PLAY0116=Copyright  
PLAY0117=Priority  
PLAY0118=Comment  
PLAY0119=Track


PLAY011A=Year  
PLAY011B=Cover  
PLAY011C=Redirect  
PLAY0120=Unknown  
PLAY0121=Video  
PLAY0122=Audio  
PLAY0123=Subtitle  
PLAB0000=Buffering  
VIDE0000=Video  
AUDI0000=Audio  
PLAY0020=Normal buffer size  
PLAY0063=Play music in background  
PLAY0072=Play movie in background  
PLAY0021=Microdrive mode  
PLAY0080=Microdrive buffer size  
PLAY0022=Repeat  
PLAY0023=Shuffle  
PLAY0024=Play at open  
PLAY007B=Play at open in fullscreen  
PLAY00B9=Exit after cmd.line playback  
PLAY0048=Keep playlist  
ADVP002A=Soft-drop tolerance  
ADVP002B=Hard-drop tolerance  
PLAY0068=Rewind step  
PLAY007C=Fast forward step  
PLAY002C=Audio output  
PLAY002D=Video output  
ADVP002C=Manual A/V offset +/-  
PLAY0067=Preload at underrun  
PLAY00BB=Preload for audio  
PLAY00A3=Microdrive starts at  
PLAY0034=Dither  
PLAY0035=Fullscreen zoom  
PLAY0036=GUI zoom  
PLAY003D=Pre-rotate portrait movies  
PLAY0040=Volume  
PLAY0041=Mute

PLAY009D=Left/Right  
PLAY009E=Preamp  
PLAY0042=Audio quality  
PLAY00BC=Video quality  
PLAY0083=Audio output  
PLAY0084=Video output  
PLAY0047=Smooth Zoom  
PLAY0044=Acceleration  
PLAY0043=Swap stereo  
PLAY0074=Stereo mode  
PLAY003F=Playing speed  
PLAY004A=Fast forward speed  
PLAY00BF=Show video in background  
PLAY00C0=Single click fullscreen  
COLO0000=Colors  
COLO0037=Brightness  
COLO0038=Contrast  
COLO0062=Saturation  
COLO0063=Reset  
COLO0064=Red offset  
COLO0065=Green offset  
COLO0066=Blue offset  
COLO2000=Mobile  
COLO2001=Color Boost  
ERR\_0000=No error  
ERR\_FFFF=Buffer is full!  
ERR\_FFFE=Out of memory!  
ERR\_FFFD=Invalid data!  
ERR\_FFFC=Invalid param!  
ERR\_FFFB=Feature not supported!  
ERR\_FFFA=Need more data!  
ERR\_FFF9=Stream added!  
ERR\_FFF8=File not found!\n%s  
ERR\_FFF7=End of file!  
ERR\_FFF6=Device error!  
ERR\_FFF5=Synchronized!  
ERR\_FFF4=Data not found!

ERR\_FFF3=%s protocol not supported!  
ERR\_FFF1=%s plugin is not compatible with current player version. Please update!  
ERR\_FFF0=Could not open URL!  
ERR\_FFED=Unauthorized request!  
ERR\_FFEB=Data read error!  
ERR\_FFEA=Data write error!  
ERR\_FFE9=Network address not found!  
ERR\_FFE8=Network not available!  
EQUA0000=Equalizer  
EQUA0180=Pre-amp  
EQUA018E=Reset  
EQUA018F=Enabled  
EQUA0190=Volume normalization  
EQPR2000=Bass Boost  
EQPR2001=Classical  
EQPR2002=Dance  
EQPR2003=Live  
EQPR2004=Pop  
EQPR2005=Rock  
EQPR2006=Soft  
EQPR2007=Techo  
EQPR2008=Treble Boost  
ASSO0000=File Associations  
PLAT0000=General  
PLAT0010=Language  
ADVP0000=Advanced  
ADVP001F=Old style toolbars  
ADVP0020=No backlight keepalive for video  
ADVP0028=No wireless MMX usage  
ADVP0029=Less rotation tearing (slower)  
ADVP002D=Home Screen time out with music playback  
ADVP0030=Benchmark from current position  
ADVP0031=Show pixel aspect ratio menu  
ADVP0036=Prefer less buffering over framerate  
ADVP0034=Use system volume  
ADVP0035=Override AVI timing based on audio  
ADVP0037=Use VR41xx tweaks

ADVP0038=D-Pad follow screen orientation  
ADVP0039=Use writeable storage memory tweak  
ADVP003A=Widcomm BT Audio button support  
ADVP003C=Disable non-critical battery warnings  
ADVP003D=No additional safe event checking  
ADVP003E=Always check SD card for plugins  
ADVP003F=Force high priority audio output thread  
ADVP0040=Auto low quality for large videos  
ADVP0041=Disable AVC deblocking filter  
ADVP0042=Blink LED when screen turned off  
ADVP0043=Capture all buttons (like games)  
PLAT0012=Model  
PLAT0011=Processor  
PLAT0022=Clock speed  
PLAT0017=Platform  
PLAT001B=OS Version  
PLAT0018=OEM Info  
ADVP0025=Slow video memory  
ADVP0026=Prefer lookup tables over arithmetic  
PLAT0027=Advanced platform settings:  
PLAT006A=OK  
PLAT006B=Cancel  
PLAT0066=Yes  
PLAT0067=No  
PLAT0082=Reset  
PLAT008E=None  
PLAT008F=Disabled  
PLAT0090=Off  
PLAT0091=On  
PLAT0092=KB  
PLAT0093=MB  
PLAT0094=sec  
PLAT0095=%  
PLAT006C=Error  
PLAT0079=Assign  
PLAT007A=Clear  
PLAT007B=Action

PLAT0073=Space  
PLAT006E=Enter  
PLAT0089=Hold  
PLAT008A=Vol. Up  
PLAT008B=Vol. Down  
PLAT008C=Tab. Next  
PLAT008D=Tab. Prev  
PLAT0088=Done  
PLAT0083=Escape  
PLAT0084=Prev  
PLAT0085=Next  
PLAT0086=Play  
PLAT0087=Stop  
PLAT0071=Left  
PLAT0072=Right  
PLAT006F=Up  
PLAT0070=Down  
PLAT0074=App %d  
PLAT0075=Ctrl  
PLAT0076=Shift  
PLAT0077=Alt  
PLAT0078=Win  
PLAT0064=Unexpected program failure. Please send "crash.txt" to the developers. Program will now exit.  
PLAT0065=Crash  
PLAT006D=Not enough memory! Free up some memory or decrease Buffer Size.  
PLAT0068=All information is dumped to "dump.xml".  
PLAT0069=Dump  
INTF0000=Hot Keys  
INTF00DD=Options  
INTF00CD=Speed  
INTF0132=Repeat  
INTF0133=Shuffle  
INTF0134=Dither  
INTF00CF=Zoom  
INTF0138=Fit Best  
INTF0234=Fit + 10%

INTF0235=Fit + 20%  
INTF0236=Fit + 30%  
INTF0237=Fit + 40%  
INTF024A=Stretch to Screen  
INTF024B=Fill Screen  
INTF00D2=Video  
INTF00D3=Audio  
INTF018D=Quality  
INTF0144=Low  
INTF0145=Medium  
INTF0146=High  
INTF0250=Quality  
INTF0251=Lowest  
INTF0252=Low  
INTF0253=Normal  
INTF0254=Preamp decrease  
INTF0255=Preamp 0 (clear)  
INTF0256=Preamp increase  
INTF0157=Disabled  
INTF0158=Disabled  
INTF017F=Fullscreen  
INTF0180=Mute  
INTF0177=Settings  
INTF00D0=Orientation  
INTF00D1=Fullscreen  
INTF0185=Left-handed  
INTF0186=Upside Down  
INTF0247=Follow GUI  
INTF0188=Normal  
INTF0189=Right-handed  
INTF00F0=Rotate by 90   
INTF0224=Toggle Zoom Fit  
INTF0225=Zoom in  
INTF0226=Zoom out  
INTF0227=Change video stream  
INTF0228=Change audio stream  
INTF0229=Change subtitle

INTF0240=Titlebar  
INTF0241=Timeslider  
INTF0242=Taskbar  
INTF018B=Video Driver:  
INTF018C=Audio Driver:  
INTF00E7=%s Decoder  
INTF00F3=Volume Up  
INTF00F4=Volume Down  
INTF022B=Volume Up Fine  
INTF022C=Volume Down Fine  
INTF00F5=Brightness Up  
INTF00F6=Brightness Down  
INTF00F7=Contrast Up  
INTF00F8=Contrast Down  
INTF0102=Saturation Up  
INTF0101=Saturation Down  
INTF00F9=Screen on/off  
INTF00DC=File  
INTF0181=Open File...  
INTF0280=CoreTheque...  
INTF017E=Playlist  
INTF014A=Media Info...  
INTF00E6=Benchmark  
INTF014B=About  
INTF014C=Exit  
INTF0172=Play  
INTF00F1=Play Fullscreen  
INTF0246=Play  
INTF0174=Stop  
INTF00FA=Move Forward  
INTF00FB=Move Back  
INTF017A=Next  
INTF0190=Next  
INTF017B=Previous  
INTF0183=Beginning/Previous  
INTF0191=Beginning/Previous  
INTF0182=Fast Forward



INTF0122=Equalizer  
INTF0200=Chapters  
INTF0223=None  
INTF0243=Subtitles  
INTF0244=None  
INTF0245=Subtitle  
INTF0201=Streams  
INTF0221=None  
INTF0202=Streams:  
INTF0222=None  
INTF0216=Channels  
INTF0203=Stereo  
INTF0140=Stereo swapped  
INTF0248=Mono Join  
INTF0204=Mono Left  
INTF0205=Mono Right  
INTF020E=View  
INTF020F=Pixel Aspect Ratio  
INTF0210=Square  
INTF0238=4:3 Screen  
INTF0211=4:3 PAL  
INTF0212=4:3 NTSC  
INTF0239=16:9 Screen  
INTF0213=16:9 PAL  
INTF0214=16:9 NTSC  
INTF0215=Auto  
INTF0260=Skin...  
INTF0261=Swap mono left/right  
INTF0505=Capture all buttons (like games)  
INTF0506=On some devices you have to turn on "capture all buttons" to this special hot key properly released after program exit (if original function doesn't work, use soft-reset)  
INTF0507=Some of the assigned hot keys may not be properly released after program exit.  
PLST0300=Title  
PLST0301=Length  
PLST0302=URL  
PLST0200=Add Files...  
PLST0201=Delete

PLST0202=Delete All  
PLST0203=Sort By Name  
PLST0204=Up  
PLST0205=Down  
PLST0206=Load Playlist...  
PLST0207=Save Playlist...  
PLST0208=File  
PLST0209=Play  
PLST020A=All  
PLST020B=None  
PLST020C=Crop  
OPEN0000=Files  
OPEN0110=Show only name  
OPEN020C=Options  
OPEN020B=Sort By  
OPEN0210=Filter  
OPEN020A=Selecting  
OPEN0211=All  
OPEN0212=Enter URL  
OPEN0213=Go  
OPEN0214=History  
OPEN0215=Empty  
OPEN0216=Name is empty!  
OPEN0217=Name  
OPEN0218=Type  
OPEN0219=Save  
OPEN021A=Enter file name  
OPEN021B=Dir  
OPEN021C=Directory  
OPEN021D=Select  
FMTM0000=Media files (all types)  
FMTL0000=Playlist files (all types)  
OPEN0201=All files (\*.\*)  
OPEN0202=Folder  
OPEN0206=Up  
OPEN0203=Name  
OPEN0204=Type

OPEN0205=Size  
OPEN020E=Date  
SETU0000=Settings  
SETU0100=Select Page  
SETU0101=Please restart the program for changes to take effect!  
SETU0102=Please warm boot the device for changes to take effect!  
SETU0103=Use the 'Select Page' menu for selecting further settings  
MEDI0000=Media Info  
MEDI0100=Format  
MEDI0101=Codec  
MEDI0102=Video Stream  
MEDI0103=Audio Stream  
MEDI0104=Frame Rate  
MEDI0106=Video Size  
MEDI0109=Data Bit Rate  
MEDI010A=Played Frames  
MEDI010B=Dropped  
MEDI010C=Played FPS  
MEDI010D=Audio Format  
MEDI010E=Mono  
MEDI010F=Stereo  
BENC0000=Benchmark Results  
BENC010B=Save Results  
BENC010E=Results are saved as "%s"  
BENC0100=Video Frames  
BENC0106=Audio Samples  
BENC0101=Bench. Time  
BENC0102=Bench. Frame Rate  
BENC0107=Bench. Sample Rate  
BENC0103=Original Time  
BENC0104=Original Frame Rate  
BENC0108=Original Sample Rate  
BENC0105=Average Speed  
BENC0110=Amount of Data  
BENC0111=Bench. Data Rate  
BENC0112=Original Data Rate  
BENC0113=Video zoom

BENC010A=Audio/Video out of sync is normal in this mode, especially in high speed situations. The point of this mode is not having any synchronization and delays.

BENC0120=%s Version %s Benchmark Results

BENC0121=URL

BENC0122=Size

BENC0123=Custom Settings

BENC0124=Some frames were dropped during benchmarking (probably invalid media file or out of memory). The results are most likely invalid!

QKEY0000=Assign Key

QKEY00C9=Press the button you want to assign!

QKEY00CA=Options

QKEY00CB=Catch in background

PLST0000=Playlist

ABOU0000=About

ABOU00CC=Dump Settings

ABOU00C8=About %s

ABOU00C9=%s Version %s

ABOU00CB=This program is FREE software and may be distributed according to the terms of the GNU General Public License.

ABOU00D6=For more information on additional licensing possibilities email [license@corecodec.com](mailto:license@corecodec.com)

ABOU00CD=Libraries used:

ABOU00CF=Special thanks!

ABOU00D1=For support or development questions and comments. Visit our community website @ <http://www.tcpmp.com>

ABOU00D3=Authors for other parts:

ABOU00D4=Copyright

ABOU00D5=Options

CTQU0000=CoreTheque

CTQU0401=Add To Playlist

CTQU0402=Reset

CTQU0403=Artist

CTQU0404=Album

CTQU0405=Track

CTQU0406=Year

CTQU0407=Genre

CTQU0408=Podcasts

CTQU0410=Playlist

CTQU0411=Location  
CTQU0412=Remove Selected

CTQU0413=Save As Playlist  
CTQU0414=Save As Query  
CTQU0420=Back From  
CTQU0421=All Of

COSP0000=Core OS Panarama files (COSP)  
FLAC0000=Free Lossless Audio files (FLAC)  
WVPF0000=WavPack files (WV)  
TTAF0000=TTA files (TTA)

NETW0000=Network  
NETW0081=Connection  
NETW0096=Use Proxy  
NETW0088=Proxy Address

NETW0092=Proxy Port  
NETP0000=Direct connection to Internet  
NETP0001=Auto-detect proxy settings  
NETP0002=Manual proxy configuration

NETF0000=28.8/36.6 Kbps Modem  
NETF0001=56 Kbps Modem/ISDN  
NETF0002=112 Kbps Dual ISDN  
NETF0003=384 Kbps DSL

NETF0004=512 Kbps DSL  
NETF0005=768 Kbps DSL  
NETF0006=1.5 Mbps T1  
NETF0007=Internat/Lan

UFDS0000=Artist  
UFDS0001=Album  
UFDS0002=Title  
UFDS0003=Genre

UFDS0004=Year  
UFDS0005=Rating

UFDA0000=All

UI\_\_0100=Now Playing  
UI\_\_0101=Open URL  
UI\_\_0102=Close

UI\_\_0103=File Selection  
UI\_\_0104=Media Properties  
UI\_\_0105=Skin  
UI\_\_0106=Buy  
UI\_\_0107=Purchase this item ?  
UIPR0205=Select All  
UIPR0206=Select None  
MENU0100=Menu  
MENU0101=Open Files  
MENU0102=Search Files  
MENU0103=Play/Pause  
MENU0104=Seek and Share  
MENU0105=Sync  
MENU0106=Fit  
MENU0107=Acceleration  
MENU0108=Properties  
MENU0109=Help  
MENU010A=?  
MENU010B=Play  
MENU010C=View  
MENU010D=Tools  
MENU010E=Preferences  
MENU010F=Colors  
MENU0110=Equalizer  
MENU0111=Custom  
MENU0112=Save Profile  
MENU0113=Delete Profile  
MENU0114=Off  
MENU0115=Hotkeys  
MENU0116=Purchase  
MENU0117=Volume  
MENU0118=Reset  
SERI0000=CoreNumber  
SERI0001=Enter Your Serial Number  
SERI0002=Incorrect Serial Number  
SERI0003=Invalid User ID. Please login to your CoreAccount with the information you received by email, and request an updated serial number.

DESK0102=Blank Screen