WINAMP 3: SKINNING TUTORIAL WEBPAGE

By: Ken Chen
Ken@winamp.com

Version 08.14.2002

TABLE OF CONTENTS

Winamp3 Skinning Tutorial

WINAMP3: SKINNING TUTORIAL WEBPAGE
by: Ken Chen

- MISSION STATEMENT:
  - The goal of this web page is to describe the skinning process to someone with graphics background and little programming experience. By creating this tutorial, we hope to generate excitement and loyalty for Winamp 3.
- KEY CONCEPTS TO CONVEY:
  - How has the skinning process fundamentally changed from Winamp 2
  - How does the graphics engine work: describe the whole process.
  - How does the layering work with XML: Alpha channels explained, Basic grouping explained.
  - How to map control to behavior using MAKI scripts within the Winamp system.
  - Encourage usage of the forum: links to the discussion
- SPECIAL THANKS TO:
  - PixelPusher for his contributions.
  - Naamloos.

Tutorial Status Report

14-August-2002
- Section 2.6 done.

30-july-2002
- Clean up Section 2-3.
- Section 4.1, 4.2 done.

25-july-2002
- Section 3.1-3.6 completed.

19-july-2002
- Complex Tutorial Skin completed.

17-july-2002
- Section 2.1-2.5 - Simple Tutorial Skin section done.

6-july-2002

- Simple Tutorial Skin completed.

Contact Us

Please email Ken Chen if you have any suggestions.

Skinning FAQ - Frequently Asked Questions

If you have any links that you think would be helpful, plea se email Ken Chen. Thanks!

Frequently Asked Questions

- I want to skin, where should i start?
- I can't open this WAL file. What's wrong?
- I can't find Skin.XML file.
- Can I get your tutorial in a single file.
- I got a skin idea. Can you make this for me?
- Is there a program that I can download that I can use to make a skin?
- Is there Winamp3 for other languages?

Skinning FAQ - Answers

I want to skin, where should i start?

I have made a tutorial on how to skin Wina mp3. You can read it here: http://www.winamp.com/nsdn/winamp3x/skinning/tutorials/.The tutorial is a work in progress. Please send me feedbacks on it or suggestions on how to make it better

---

I can't open this WAL file. What's wrong?

Don't try to open WA L file with Notepad or other text editing program. You need to do the following:
1) Rename your skin file from .WAL extension to .ZIP extension.
2) Use Winzip (http://www.winzip.com) to extract the files from inside.
3) Edit the files

I can't find Skin.XML file.

You need to extract the file from your skin (the WAL file). Here's what you need to do:

1) Rename your skin file from .WAL extension to .ZIP extension.
2) Use Winzip (http://www.winzip.com) to extract the files from inside.
3) Edit the files

Can I get your tutorial in a single file?

The tutorial site is not done yet. We're still adding more sections to it. Once it's complete, we'll make a downloadable version (probably PDF). Please check back later.

I got a skin idea. Can you make this for me?

Please post this question/Request in the forums. There are people in the community who can help you. http://forums.winamp.com/forumdisplay.php?s=&forumid=123.

I have made a tutorial on how to skin Winamp3. You can read it here: http://www.winamp.com/nsdn/winamp3x/skinning/tutorials/.The tutorial is a work in progress. Please send me feedbacks on it or suggestions on how to make it better.

Is there a program that I can download that I can use to make a skin?

We currently do not have a skin au thoring tool. I have made a tutorial on how to skin Winamp3. You can read it here: http://www.winamp.com/nsdn/winamp3x/skinning/tutorials/.The tutorial is a work in progress. Please send me feedbacks on it or suggestions on how to make it better.

Is there Winamp3 for other languages?

We're working on International Language support for Winamp3. Please check for this feature in future versions of winamp3.

Winamp3 Skinning Glossary

This page contains definitions for Winamp3 Skinning terminology. You can look up terms that are foreign to you, thereby understanding and adopting those terms. If you want any term defined, please email Ken Chen. Thanks!

Frequently Asked Questions

Graphics	
PS:	Short for Photoshop
PSP:	Short for PaintShop Pro
Alpha Channel:	An alpha channel represents the transparency information on a per -pixel basis. You can read more here: http://www.w3.org/pub/WWW/TR/PNG-DataRep.html#DR.Alpha-channel
Interlace:	Interlacing produces a "layering effect" so that a grainy version of the image is sent first, and on subsequent passes is "resolved" into the final image. This process allows people to get an idea of wha t the image contains before it is fully loaded.
PNG:	Short for Portable Network Graphics.
XML	
XML:	Extensible Mark-up Language. Used as the base for the winamp3 abstraction layer.
XML Hierarchy:	The manner in which XML tags may be nested within one ano ther to form a hierarchical, recursive 'tree' structure, with parent tags enclosing multiple child tags, which may themselves be parent tags as well. See XML section for example.
Root File:	skin.xml, this is where the interaction with xml starts.
Studio.xnf File:	the file with all your preferences and last known state.
X&Y Coordinates:	As pertaining to layout code: position of the element in the layout, X being distance from the left and Y being Distance from the top.
Winamp3 Terminology	
Elements:	Defining an element within a skin to be used later. Usually linking a bitmap id to a picture or text or fonts, etc.
Bitmap ID:	The identification given to an element in the elements file and the id used in layout for said element.
Group:	A set of objects tha t can be referred to and manipulated as a whole. It first must be defined by a <groupdef> tag, then located with a <group> tag.
Container:	A container represents a window. It is an object that holds multiple related layouts.
Component:	In global winamp t erms, a component is an advanced form of plugin, which can provide nearly any kind of new functionality to the core system. In skinner's terms, a component is the xml identity which represents the interface for the actual component (plug -in). Not all components will necessarily provide an obvious (or any) xml identity (cnv_dshow.wac, for instance)

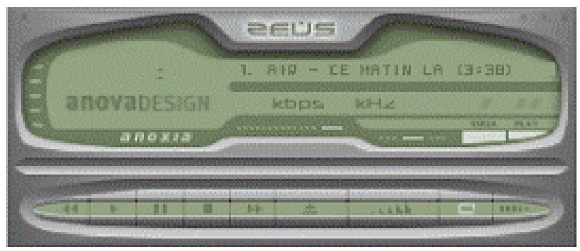
Layout:	A special kind of group, which shown inside a container. You can define multiple layouts for a container but only one layout can be visible at a time. An example is the normal mode and windowshade mode in the Default skin.
Gamma:	Most common activity for a skinner is to adjust the color of an element to "just right" to achieve the desired effect. While it is possible to adjust an element's color using a graphics program, Winamp 3 allows you to link an element to a gamma control and change them in the preference menu.
Gamma controls:	Define a gamma control so the color of that gamma group can be changed in the preference menu.
Layer:	A layer is basically an image. Layers can overlay on top of each other. It is similar to a layer in Adobe Photoshop.
MAKI:	**MAKI** stands for **M**ake **A K**iller **I**nterface.
M file:	An uncompiled Maki file containing the readable source code
Maki File:	A compiled M file, also known a s a script.
Maki Monkey:	Someone who knows how to write M files well.
Drawer:	A Drawer is an animated layer that is normally hidden but can be shown when triggered by a user event (often a mouse click).

Part 0. Intro to Winamp3

- Why do we need Winamp3?
- Winamp3 Components

Why do we need Winamp3?

- Limitations of Winamp 2.x
  - Skins only consist of static bitmap replacement. The app cannot be deployed with radically different interfaces or behaviors.
  - Plugins cannot publish interfaces to each other, thereby limiting their ability to leverage one another.



  - Winamp 2.x was not built with portability in mind.

Winamp3 Components?

Features of Winamp3:

- Skin system:
  - XML-based format
  - freeform
  - scripting
  - emulation of Winamp 2.x skins
  - on-the-fly bilinear filtered scaling of any winamp piece
  - animations, realtime transformations
  - alpha transparency
  - color correction of any skin
- Component architecture:
  - allows any kind of plugin you can think of
  - components become part of the system and extend it
  - components can interoperate without even knowing who they are talking to
  - component windows and interfaces are skinnable
  - components are built with the same C++ library as the main product for consistency and to enable developers to quickly create powerful addons
  - using a dispatchable interface to make interaction of multiple plugin versions easier
- New audio playback system:
  - crossfading
  - "dsp" plugins can be stacked
  - gapless output (even for cd playback)

- Playlist:
  - XML-based format
  - ability to queue up files in playlist out of order (hold down alt)
  - Media library
- Database:
  - caching
  - queries
  - multiple indexes and views on tables

Part I. Winamp 2 to Winamp3

OUTLINE

Transitioning from Winamp 2 and venturing into the new world of Winamp3!

- Basically a quick start for Winamp 2 Pros
- Purpose
    - Explain how has the skinning process fundamentally changed from Winamp 2
    - Difference in the engine and philosophy
- Highlighting the key areas to consider and emphasizing the new dynamic nature of the player.
- Include/Link to: an overview of new components with the new Winamp3. Maybe doing a comprehensive list of all the Components (top to bottom) would be helpful.
- Maybe even a definition/glossary of all the components along with a detailed outline of the functionality of each component would also come in handy. Perhaps a FAQ. An example would be a definition of what the "Thinger" is and what it functionality is and maybe even to go so far as to what it's limitations are.

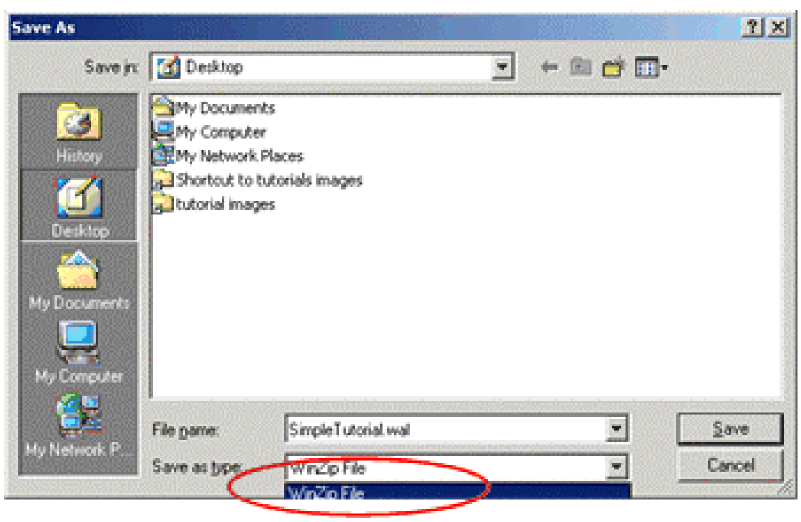
W3 Skinning Tutorial - 2.1 Start Here!

In section 2, I will show how to make a simple skin and teach you the basic Winamp3 In section 2, I will show how to make a simple skin and teach you the basic Winamp3 terminologies. Before you start, you will need these items.

Simple Tutorial Skin

- We're going to show you how to make a simple skin like the one shown below. Download the "Simple Tutorial Skin" example (http://download.nullsoft.com/winamp/skin-dev/SimpleTutorial.wal)

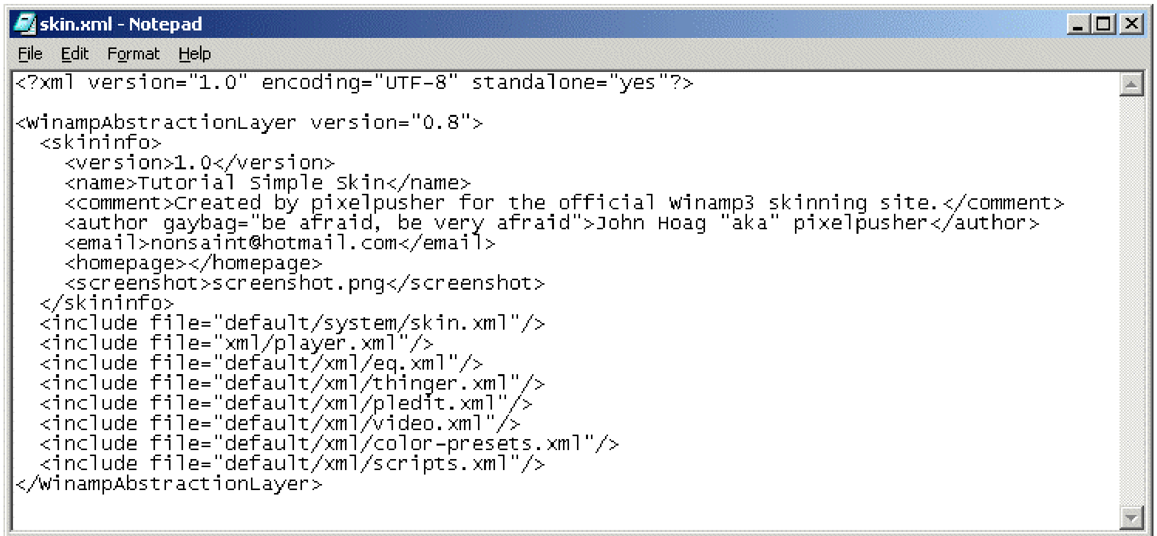
- When you save the file, make sure that the type of the file is selected to "All Type" and **NOT** "Winzip" filetype. You should save the file into your skin subdirectory in your Winamp3 directory. For most people, that path is "C: \Program Files\Winamp3\Skins".



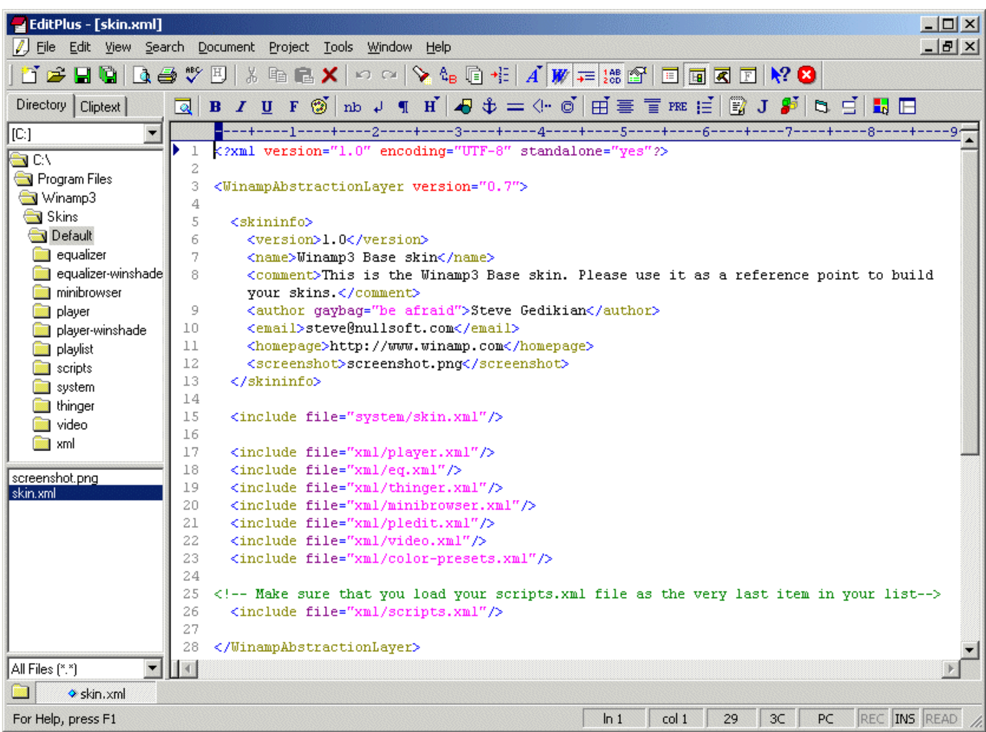
Text Editing Programs

You can use any text editing program to write XML and MAKI script files.

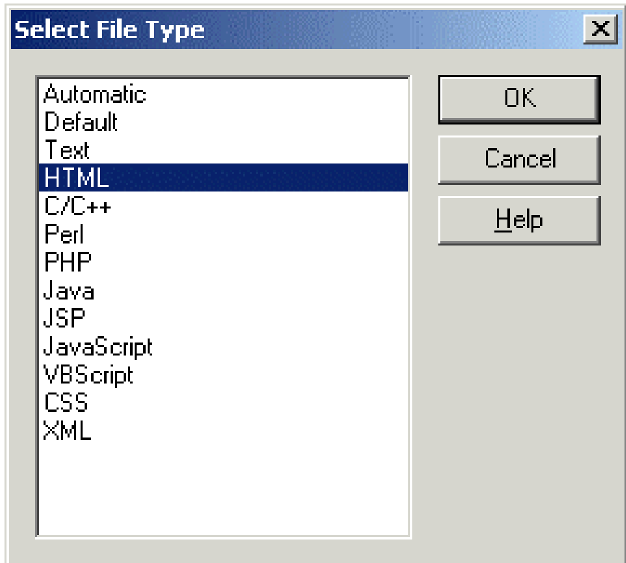
1) The plain Windows Notepad will do. It is under "Start Menu" => "Programs => "Accessories => "Notepad". Here's what it looks like.



2) However, if you intend to do alot of coding rather a quick editing, we recommend you to use something like EditPlus. EditPlus will automatically color your HTML and XML code. The coloring makes it easier for you to see your code and find missing closing tags. Here's what it looks like.

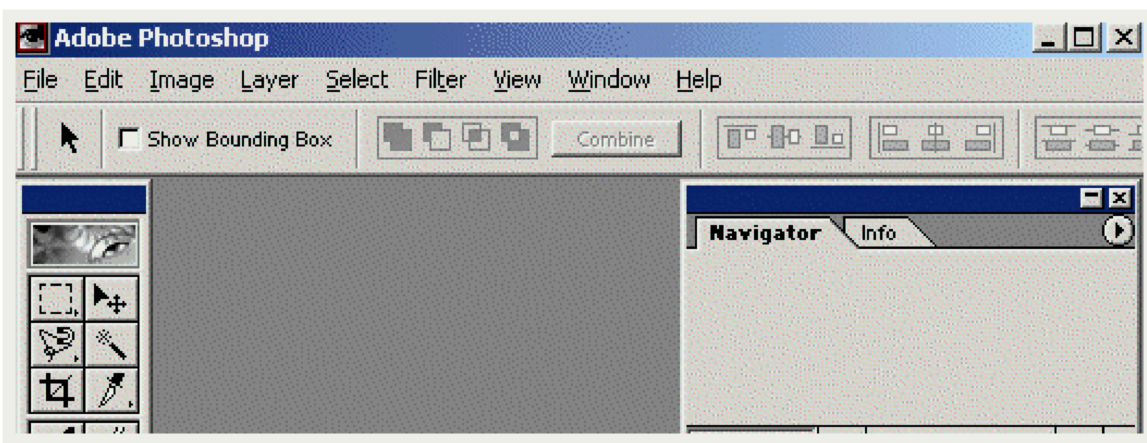


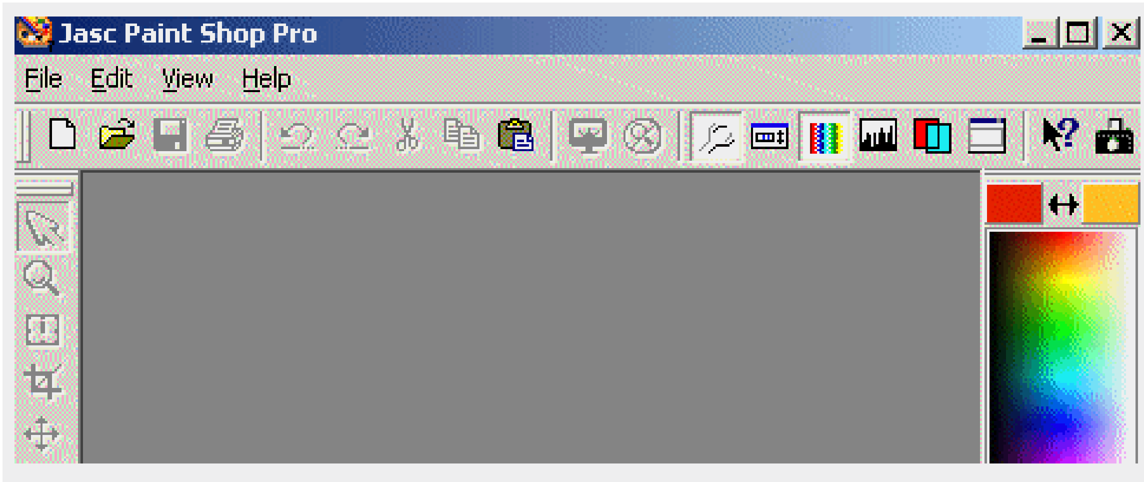
You can select the type of coloring by going to "Document" menu => "Change File Type..."



A graphics program

We recommend Adobe Photoshop (www.adobe.com) or Paintshop Pro (www.psp.com). Here're some tutorials (http://forums.winamp.com/showthread.php?s=&threadid=93803) if you're new to graphics.





Contact Us

Please email Ken Chen if you have any suggestion.

W3 Skinning Tutorial - 2.2 XML Introduction

In this section, I will give a short introduction to XML. You may ask: what does XML have to do with Winamp? Well, XML is fully implemented in Winamp. From the skinning files to the playlist, it's all XML based. I will go over XML basics in this section. If you know XML, go ahead and skip to the next section.

In this section, I will discuss:
1. So what is XML?
2. XML Hierarchy
3. XML & Winamp

1. So what is XML?

XML stands for *eXtensible Markup Language* (**XML**). You hear about XML everywhere. We're going to give you a quick and dirty guide to XML and its relevance to Winamp. XML is just like HTML. If you know HTML, you can pick up XML very quickly. So don't be afraid.

HTML and XML are from the same family of what is called **SGML** (*Standard Generalized Markup Language*). SGML was invented as a text-based language that can be used to **mark up** data - that is, add metadata - in a way which is **self describing**. What does this mean? Basically SGML is a way for programmers to transfer and display information. The best known application of SGML is *HyperText Markup Language*, or **HTML**. HTML is a universal markup language for the display of information and the linking of different pieces of information. The idea was that any HTML document (or **web page**) would be presentable in any application that was capable understanding HTML (namely a **web browser**).

Although HTML has been incredibly successful, it's also limited in its scope: it is only intended for displaying documents in a browser. It is not particularly useful as a medium to convey information to other programs. Don't understand? Here let me give you an example.

For example, if I want to tell everyone my name, there are many ways I can do this:

a) I can create a text file on a computer that look like this. When I send you this text file, you can read it or write a program to parse it. By reading "Ken Chen" in the text file, you can guess that it's my name. But then again, are you sure? It's not really clear, right? Lets look at the next example.

```
Ken Chen
```

b) For the web, I can create an HTML file like this. I can put it on a web site and you can see the page in your browser. Once again, by just reading "Ken Chen" you're not really sure that it's my name.

```
<HTML>
 <HEAD><TITLE>Name</TITLE></HEAD>
 <BODY>
  <P>Ken Chen</P>
```

```
  </BODY>
</HTML>
```

c) XML is a standard for conveying information. I might create an XML file like this. Looking at this file, you can see that it looks very much like an HTML file. The style is very similar. The difference is that in XML, you can design or come up with any tag you want to convey the information you want. Reading this simple example, you can tell that my first name is Ken and my last name is Chen.

```
<Name>
  <First>Ken</First>
  <Last>Chen</Last>
</Name>
```

From this simple example you can see why markup language like SGML and XML are called self - describing. Looking at the data, you can easily tell that this is information about a <name> and you can see that there is data called <first> and another data called <last>.

d) You could write a shorter version of the previous XML by using element attributes. I ike so:

```
<Name First="Ken" Last="Chen" />
```

and it could equate to the one in C, if the Name element had those attributes (first and last). The reason for this is that you get tired of typing open and closing tags. Once you open a tag, you can have attributes for that tag. Many Winamp3 skin XML files are in this style. Here's an example from the Default skin.

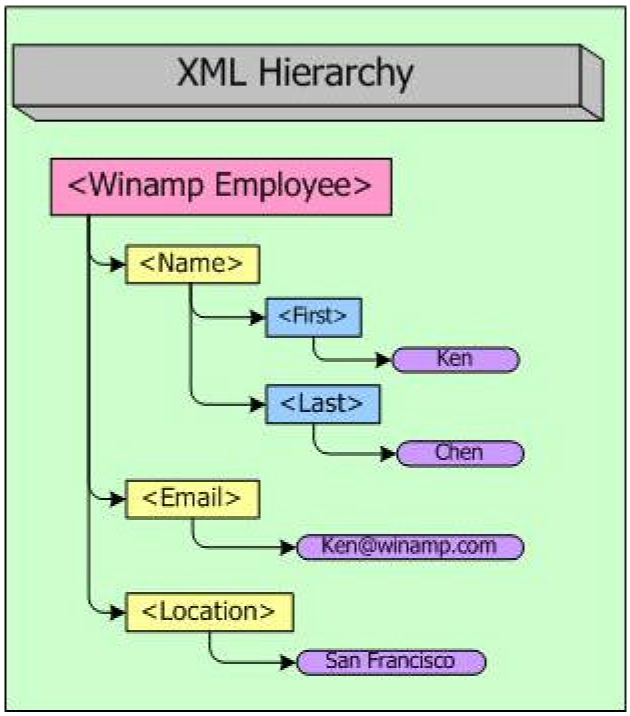
```
<button
  id="Previous"
  action="PREV"
  x="0" y="0"
  image="player.button.previous"
  downImage="player.button.previous.pressed"
  tooltip="Previous"
/>
```

2. XML Hierarchy

The beauty of XML is that you can design whatever format you want to describe your data. The key is that you follow the XML hierarchy (we'll describe in the next section. Other than that, make sure you close the tag when you one. (ie: make sure you have </name> if you open a <name> tag.

XML has a hierarchy that you need to follow. This is much like the HTML code where you embed a tag within a tag. By doing that, you're creating another level in the hierarchy. Make sure you close your tags correctly. Check out this example.

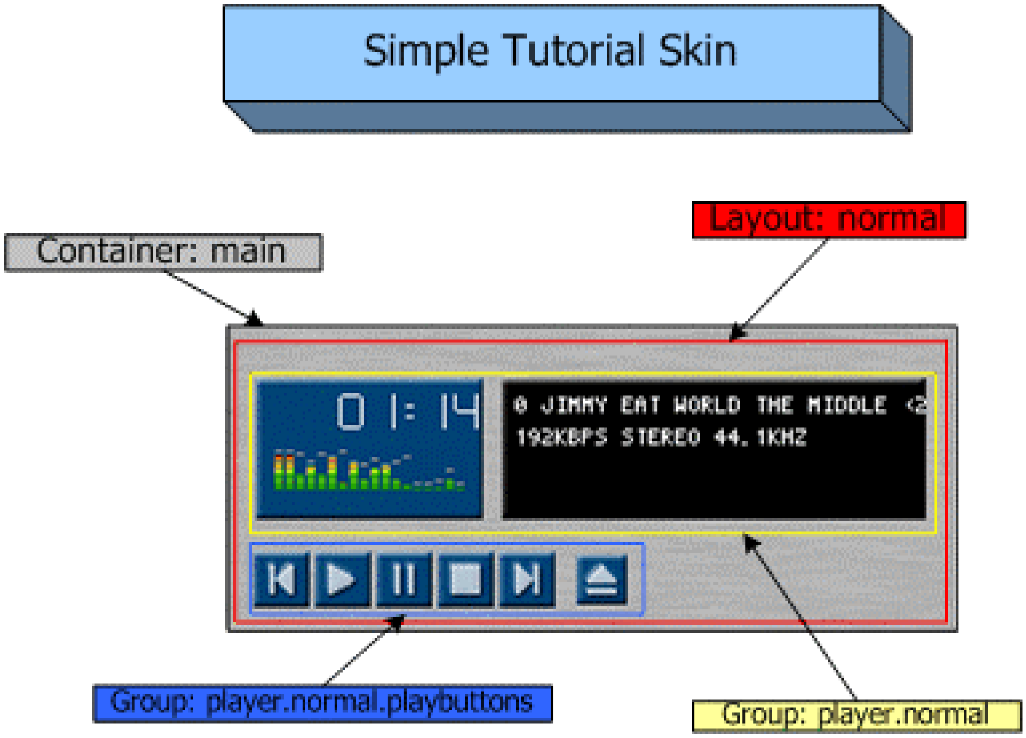
```
<Winamp Employee>
 <Name>
  <First>Ken</First>
  <Last>Chen</Last>
 </Name>
 <Email>Ken@Winamp.com</Email>
 <Location>San Francisco, CA</Location>
</Winamp Employee>
```



In this example, we have <Name>, <Email> and <Location> as level 1 tags. Within the <Name> tag, we have two level 2 tags: <First> and <Last>.

3. XML & Winamp

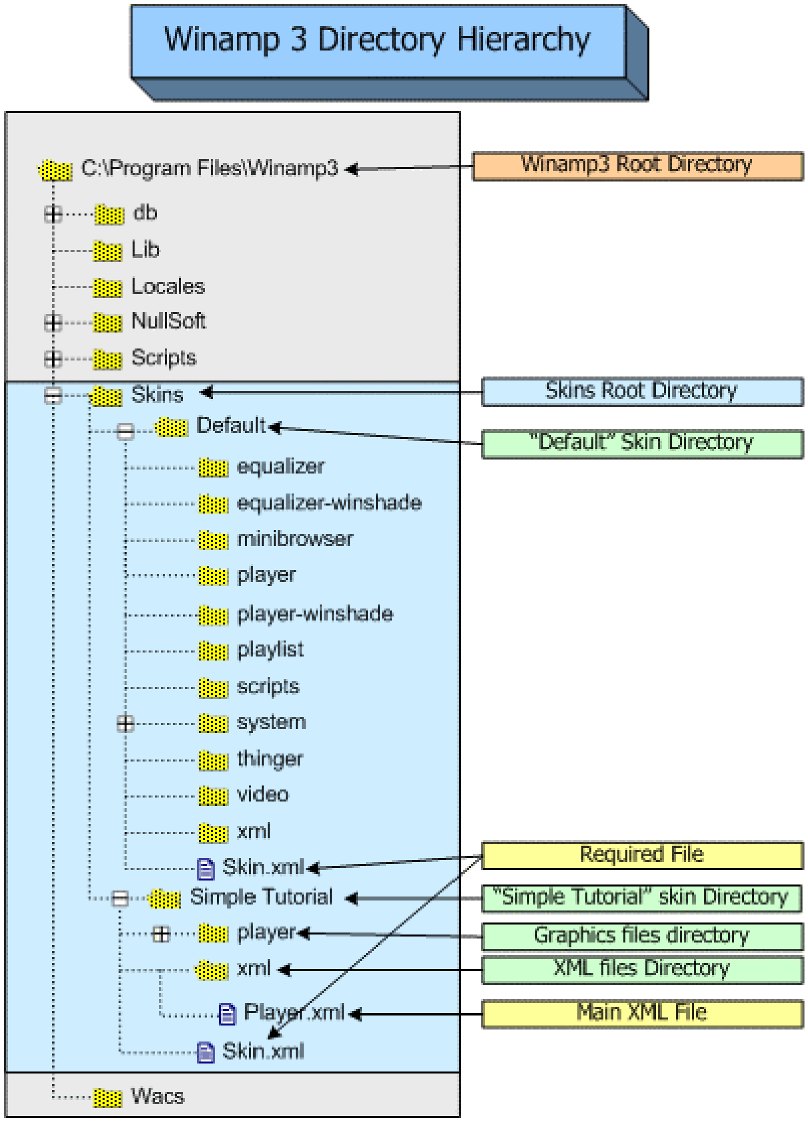
XML is basically a way to store data. Now we've shown you what XML is, you may ask: what does that have to do with Winamp? Well, XML is fully implemented in Winamp. From the skinning files to the playlist, it's all XML based. I've gone over the basics of XML. I'll go over Winamp XML elements and dissect apart a simple skin example in the next section.



We will begin with a simple skin example. I'll give you the essential parts of the skin. In the next section, I'll demonstrate important Winamp3 concepts using this skin. The topics that I'll go over are:

1. Winamp3 Skin Directory Hierarchy
2. Simple Tutorial Skin - Graphics Files
3. Simple Tutorial Skin - Skin.xml
4. Simple Tutorial Skin - Player.xml

1. Winamp3 Skin Directory Hierarchy



a) Skin Hierarchy

Before you start playing with a skin, you hav e to know where everything is. Above is a picture of the Winamp3 directory. If you open that directory ("C: \Program Files\Winamp3") with Windows Explorer, it should pretty similar. The section in blue is the "Skin" directory and the directory that we care about. Within the "Skin" directory, there are sub -directories and files. Each sub-directory and WAL files are a skin. WAL is Winamp3 skin format. Next section talks about how to manipulate it.

b) .WAL files

Instead of a directory for a skin, you may see only one file with a .WAL extension. Don't worry.

WAL (Winamp Abstraction Layer) is Winamp3 skin format. It is basically a zipped file of the skin directory. We do this to save space and to make it easier to distribute the skins. To make changes to that skin, do the following:

1) Rename the .WAL file to .ZIP
2) Open that file with Winzip. If you don't have it, you can get it here. www.winzip.com
3) Unzip the content into a directory.

c) Required File

Lets take a look at what's in a skin directory. We'll use the "Simple Tutorial" as an example. Keep in mind that other skins may contain other files and directories that are not shown. What we're showing here is the bare minimum example.

Within the "Simple Tutorial" skin directory, there are three items: 1) Skin.xml file, 2) "Player" directory, 3) "XML" directory. The only required file for a skin is the Skin.xml file. It is the file that the Winamp3 skin engine looks for. You can say that it is the entry point for t he skin. From that file, all the fun starts. =) I will dissect that file later on.

The "Player" directory contains all the graphics file for the skin. The files are in .PNG (Portable Network Graphics) format. The "XML" directory is usually where all the X ML files are located. Because our skin is relatively simple, there is only one file in that directory.

d) #include in XML files

One more thing I want to go over is the <include file="xml/player.xml"/ > tag. When the Winamp skinning engine parses this XM L file and encounters that tag, it grabs the file specified within that tag and paste whatever code that is inside that file into the current file. So in this example, we're looking at the skin.xml file. The skinning engine will grab player.xml file in the xml subdirectory and paste whatever code that is inside player.xml into skin.xml. This process is recursive. This means that if there's any include file tag inside player.xml, it will get those code also.

2. Simple Tutorial Skin - Graphics Files

These are the graphics that I've used in creating Simple Tutorial skin. One thing to notice is that we like to use PNG file format. Please also use that format too. The graphics files are the parts that we're going to use for our skin. We will define the m as elements and link them into our XML code later in order to use them.

3. Simple Tutorial Skin - skin.xml

SKIN.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<WinampAbstractionLayer version="0.8">
  <skininfo>
    <version>1.0</version>
    <name>Tutorial Simple Skin</name>
    <comment>Created by pixelpusher for the official Winamp3 skinning site.</comment>
    <author gaybag="be afraid, be very afraid">John Hoag "aka" pixelpusher</author>
    <email>nonsaint@hotmail.com</email>
    <homepage></homepage>
    <screenshot>screenshot.png</screenshot>
  </skininfo>
  <include file="@DEFAULTSKINPATH@xml/eq.xml"/>
  <include file="@DEFAULTSKINPATH@xml/thinger.xml"/>
  <include file="@DEFAULTSKINPATH@xml/pledit.xml"/>
  <include file="@DEFAULTSKINPATH@xml/player-elements.xml"/>
  <include file="@DEFAULTSKINPATH@xml/pledit-normal-group.xml"/>
  <include file="@DEFAULTSKINPATH@xml/video.xml"/>
  <include file="xml/player.xml"/>
</WinampAbstractionLayer>
```

LINE NUMBER CODE

The required file for any skin is the Skin.xml file. You can locate this file in your "C: \Program Files\Winamp3\Skins\Simple Tutorial" directory. The content of that file is posted above. I'll go over all the parts. I have listed line number on the left and the code on the right. You can open your version and follow along.

a) <?XML> (line 1)

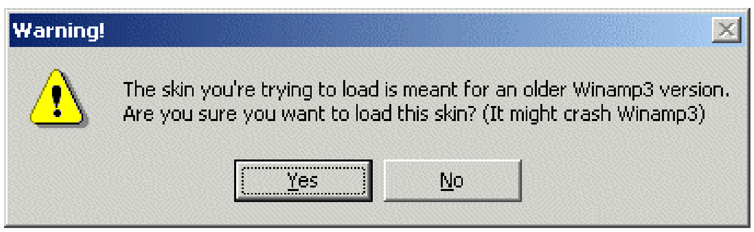
```
0001 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

The first line of the Skin.xml file should be the line shown above. The line identifies Skin.xml as an XML file. It is important to remember that you only that line only once in Skin.xml. If you divide your xml into several files, do not include that line in other xml files. Subsequent xml files wi ll be pasted into Skin.xml file by the skin engine.

b) <WinampAbstractionLayer> (line 3)

```
0003 <WinampAbstractionLayer version="0.8">
```

The WinampAbstractionLayer tag is used to specify which skin version you're following. The version that "Simple Tutorial" skin is made for is 0.8. This tag is use d to track compatibility. We will keep compatibility as long as possible. But sometimes when we introduce a new functionality and change an existing one, compatibility is broken and we will increment the version number. You will need to update the skin to conform to the new standard. If the skin engine finds your version number is different than the current version, you will see the warning below. However, this does not mean that your skin will no longer work. If functionality that has been changed is not implemented in your skin, you skin will continue to function.



c) <skininfo> section (line 4-12)

```
0004 <skininfo>
0005   <version>1.0</version>
0006   <name>Tutorial Simple Skin</name>
0007   <comment>Created by pixelpusher for the official Winamp3 skinning site.</comment>
0008   <author gaybag="be afraid, be very afraid">John Hoag "aka" pixelpusher</author>
0009   <email>nonsaint@hotmail.com</email>
0010   <homepage></homepage>
0011   <screenshot>screenshot.png</screenshot>
0012 </skininfo>
```

The tags are all pretty self-explanatory. But lets go over them anyways for fun.

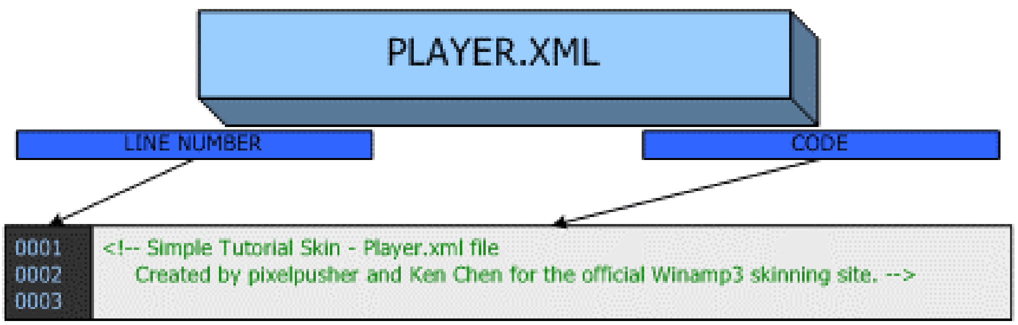
0004	<skininfo>	
0005	<version>	- the version of the skin
0006	<name>	- the name of the skin.
0007	<comment>	- comments for the skin
0008	<author>	- author's name
0009	<email>	- email of the author
0010	<homepage>	- homepage for the skin. May be used for user to go download an updated version of the skin.
0011	<screenshot>	- screenshot of the skin for the Skin Switch component to display.
0012	</skininfo>	

d) <include> section (line 13-20)

I have discussed <include> functionality above. Note that we're borrowing skins from the Default skin. The only component's appearance we're overriding is the main player. The specification for that is in player.xml (line 14). We'll go over that file next.

4. Simple Tutorial Skin - player.xml

a) Name and comments (line 1-3)



The first section of your XML should be a comment section that briefly explains the functionalities of the file. This way it will be easy for others to read and understand your code.

b) Elements definition (line 4-18)

```
0004 <!-- Element definition -->
0005 <elements>
0006   <bitmapfont id="player.BIGNUM" file="player/numfont.png" charwidth="9"
         charheight="13" hspacing="3" vspacing="1" gammagroup="Fonts"/>
0007   <bitmapfont id="wasabi.font.default" file="player/default-font.png" charwidth="5"
         charheight="6" hspacing="0" vspacing="0" gammagroup="Display Fonts"/>
0008   <bitmap id="player.main.background" file="player/background.png"
         gammagroup="Backgrounds"/>
0009   <bitmap id="vis.backlayer.blue" file="player/blue-vis-layer.png"
         gammagroup="Backgrounds"/>
0010   <bitmap id="black.backdrop" file="player/black-backdrop.png"
         gammagroup="Backgrounds"/>
0011   <bitmap id="blue-player.button.previous" file="player/blue/blue-previous.png"
         gammagroup="Buttons"/>
0012   <bitmap id="blue-player.button.play" file="player/blue/blue-play.png"
         gammagroup="Buttons"/>
0013   <bitmap id="blue-player.button.pause" file="player/blue/blue-pause.png"
         gammagroup="Buttons"/>
0014   <bitmap id="blue-player.button.stop" file="player/blue/blue-stop.png"
         gammagroup="Buttons"/>
0015   <bitmap id="blue-player.button.next" file="player/blue/blue-next.png"
         gammagroup="Buttons"/>
0016   <bitmap id="blue-player.button.eject" file="player/blue/blue-eject.png"
         gammagroup="Buttons"/>
0017 </elements>
0018
```

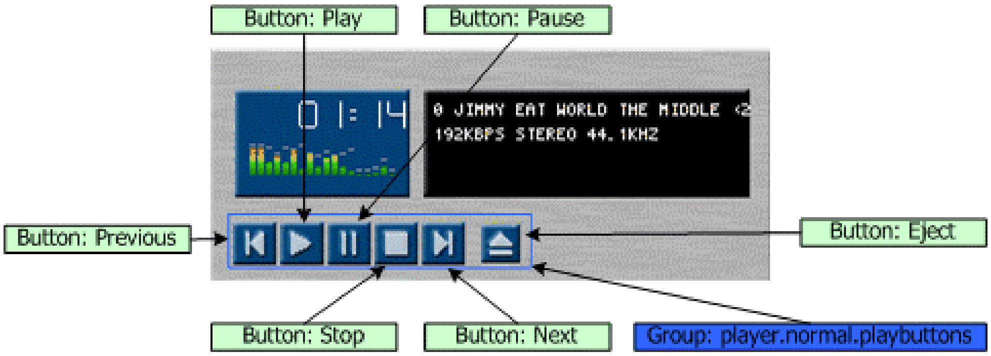
This section defines all the elements in your skin. This is the part that links the graphics to a "bitmap id" that other XML code can refer to. For example, line 11 links a bitmap id, "blue-player.button.previous", to a graphics component in "player/blue/blue-previous.png" file. That bitmap id is also linked to a gamma group called "Buttons". After linking that object to a gamma group, user can adjust the color variance of that object in the preference menu. Therefore, even if that object is blue right now, you can change it so that it'll appear to be a different color.

As your skin becomes more complex, you will have many elements defined in your skin. It will be useful to break the <Elements> definitions into different files. Typically, one would define all the elements in each component in separate files. For example, elements in the main player will be in "player-elements.xml" and elements in the playlist will be in "pledit-elements.xml".

c) Playback Buttons Group Definition (line 19-64)

```
<!-- Playback Buttons Group Definition -->
<groupdef id="player.normal.playbuttons" name="Player buttons">
 <button
  id="Previous"
  action="PREV"
  x="0" y="0"
  image="blue-player.button.previous"
  tooltip="Previous"
 />
 <button
  id="Play"
  action="PLAY"
  x="23" y="0"
  image="blue-player.button.play"
  tooltip="Play"
 />
 <button
  id="Pause"
  action="PAUSE"
  x="46" y="0"
  image="blue-player.button.pause"
  tooltip="Pause"
 />
 <button
  id="Stop"
  action="STOP"
  x="69" y="0"
  image="blue-player.button.stop"
  tooltip="Stop"
 />
 <button
  id="Next"
  action="NEXT"
  x="92" y="0"
  image="blue-player.button.next"
  tooltip="Next"
 />
 <button
  id="Eject"
  action="EJECT"
  x="121" y="1"
  image="blue-player.button.eject"
  tooltip="Eject"
 />
</groupdef>
```

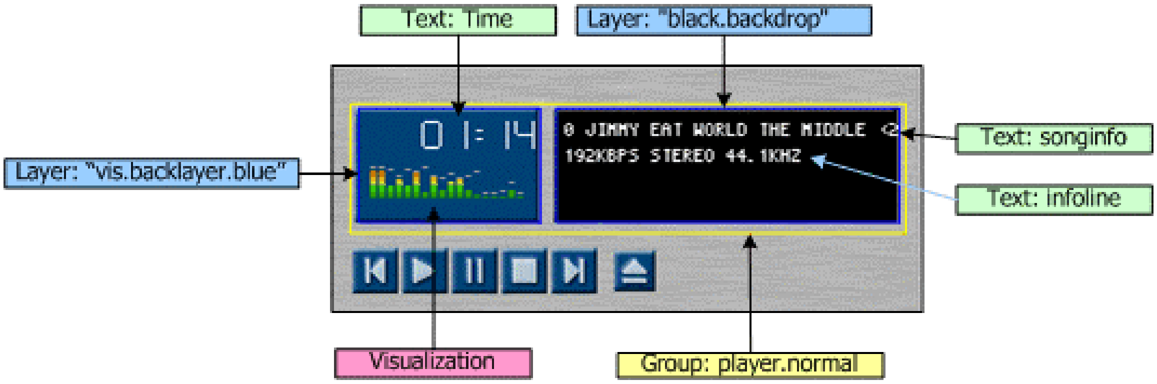
Sometime a group of objects naturally belong together or are located next to each other. If you move one of them, chances are that you'll be moving the rest of them. The playback buttons is an excellent example. If you want to change the location of those button, rather than moving each element independently, you can define a group and move the group together. A group is a set of objects that can be referred to and manipulated as a whole. It is a like a bag to throw your stuff into. It first must be defined by a <groupdef> tag, then can be located with a <group> tag. In section 2.5, we're going to dissect it even more.



d) Song File Information and Visualization Group Definition (line 65-107)

```
0065 <!-- Song File Information and Visualization Group Definition -->
0066 <groupdef id="player.normal" name="Winamp">
0067  <layer id="vis.backlayer.blue"
0068   x="13" y="20"
0069   ghost="1"
0070   image="vis.backlayer.blue"
0071  />
0072  <text
0073   id="timer"
0074   display="time"
0075   x="30" y="24"
0076   w="70" h="18"
0077   font="player.BIGNUM" fontsize="21" align="right"
0078   timecolonwidth="7"
0079  />
0080  <layer id="black.backdrop"
0081   x="105" y="20"
0082   ghost="1"
0083   image="black.backdrop"
0084  />
0085  <text
0086   id="songticker" ticker="1"
0087   display="songname" default="" showlen="1"
0088   x="108" y="23"
0089   w="157" h="14"
0090   font="wasabi.font.default" fontsize="13"
0091  />
0092  <!-- Song File Information-->
0093  <text
0094   id="infoline" ticker="1"
0095   display="songinfo" default=""
0096   x="108" y="37"
0097   w="157" h="11"
0098   font="wasabi.font.default" fontsize="12" ghost="1"
0099  />
0100  <!-- Visualzation -->
0101  <vis
0102   id="vis"
0103   x="20" y="46"
0104   w="72" h="16"
0105  />
0106 </groupdef>
0107
```

SongInfo group is another group definition just like the Playback Buttons group. As you can see that the elements defined inside this group is different from the previous one. In this example, we're using elements other than buttons. We're using the layer to set the background color of two areas. A layer is basically an image. Layers can overlay on top of each other. It is sim ilar to a layer in Adobe Photoshop. "Songinfo" and "infoline" are text objects displaying information stored within Winamp3 runtime variables.



e) Main Window: Container and Layout (line 1-3)

```
<!-- Main Window: Container and Layout -->
<container id="main" name="Main Window" default_x="0" default_y="0"
default_visible="1">
  <!-- First Layout -->
  <layout id="normal" background="player.main.background" droptarget="pldr">
              <!-- Using Playback Buttons Group -->
              <group id="player.normal" />
              <group id="player.normal.playbuttons" x="12" y="85" />
  </layout>
</container>
```

Previous sections are all about defining objects. Now we get to start using them. If you read the code, you'll see that there is a container object, a layout object and two group objects. Container is at the top of hierarchy because everything else "lives" in it. Within the container, we have defined a layout an d, within the layout, we're using the groups we defined earlier.

Container, layout and group are important concept and we will go more in depth in subsequent sections. Read on....

W3 Skinning Tutorial - 2.4 Container

What is a container?

A container is a top level object and it basically represents a window. Nothing holds a container. It is an object that holds multiple related layouts. Each layout represents an appearance for that window. You can design different layouts for each window but only o ne can be visible at a time.

```
<!-- Main Window: Container and Layout -->
<container id="main" name="Main Window" default_x="0" default_y="0"
default_visible="1">
.............................
</container>
```

Breakdown of the code:	
id="main"	The id name that others can refer to.
name="Main Window"	The name to display
default_x="0"	The default x-coordinate for the container.
default_y="0"	The default y-coordinate for the container.
default_visible="1"	Make this container visible.

Layout

What is a layout?

A layout is a special kind of group, which shown inside a container. Each layout represents an appearance for that window. Layouts give you the ability to design different looks for the same container (or window). However, only one layout can be visible at a time. You must toggle among layouts you defined. An example is the normal mode and windowshade mode in the Default skin.

- The top image is using normal layout and the bottom is using t he windowshade layout.

Lets look at the Simple Tutorial xml code for layout:

```
<!-- Main Window: Container and Layout -->
<container id="main" name="Main Window" default_x="0" default_y="0"
default_visible="1">
  <!-- First Layout -->
  <layout id="normal" background="player.main.background" droptarget="pldr">
              <!-- Using Playback Buttons Group -->
.............................
  </layout>
</container>
```

Breakdown of the code:	
id="normal"	- The id name that others can refer to.
background="player.main.background"	- The graphics file to use for the background
droptarget="pldr"	- "droptarget" redirects the event when you drag and drop a mp3 file into the container. In this case, the event is redirected to the playlist editor.

W3 Skinning Tutorial - 2.5 Group

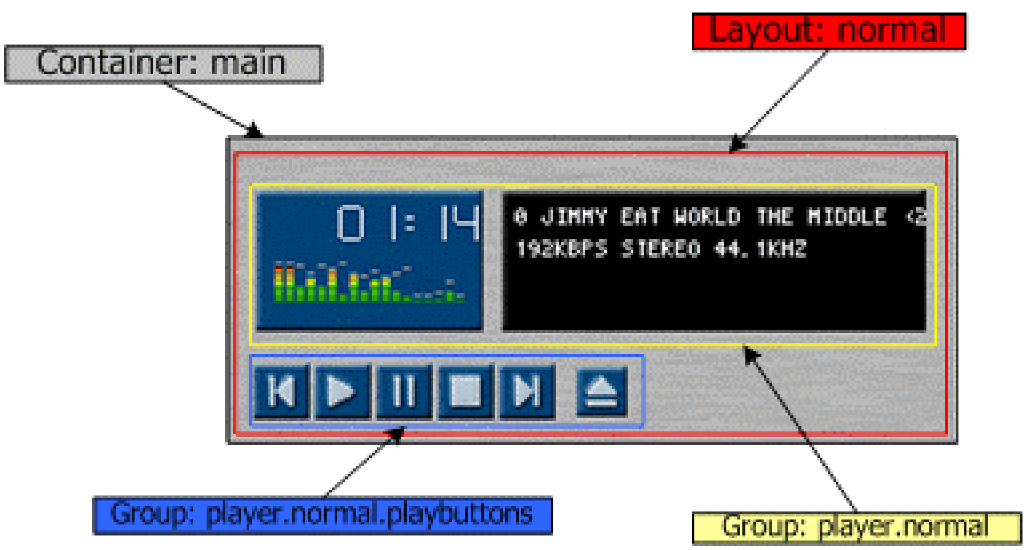
1. What is a group?
2. Simple Tutorial: Using groups
3. Simple Tutorial: Playback Buttons Group Definition
4. Simple Tutorial: Song File Information and Visualization Group Definition

1. What is a group?

A group is a set of objects that can be referred to and manipulated as a whole. First must defined by a <groupdef> tag, then located with a <group> tag. Group is an important c oncept to understand because a group is such a versatile object.

2. Simple Tutorial: Using groups

Let go back to our "Simple Tutorial" example. As you recall, we have two groups defined: "player.normal.playbuttons" and "play.normal".

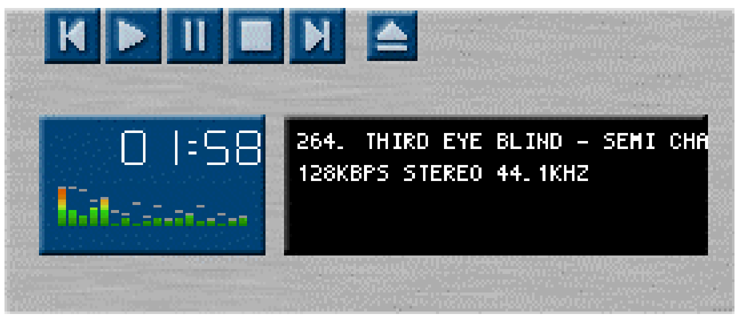


Lets take a look at the code: (player.xml)

```
0108 <!-- Main Window: Container and Layout -->
0109 <container id="main" name="Main Window" default_x="0" default_y="0"
     default_visible="1">
0110   <!-- First Layout -->
0111   <layout id="normal" background="player.main.background" droptarget="pldr">
0112            <!-- Using Playback Buttons Group -->
0113            <group id="player.normal" />
0114            <group id="player.normal.playbuttons" x="12" y="85" />
0115   </layout>
0116 </container>
```

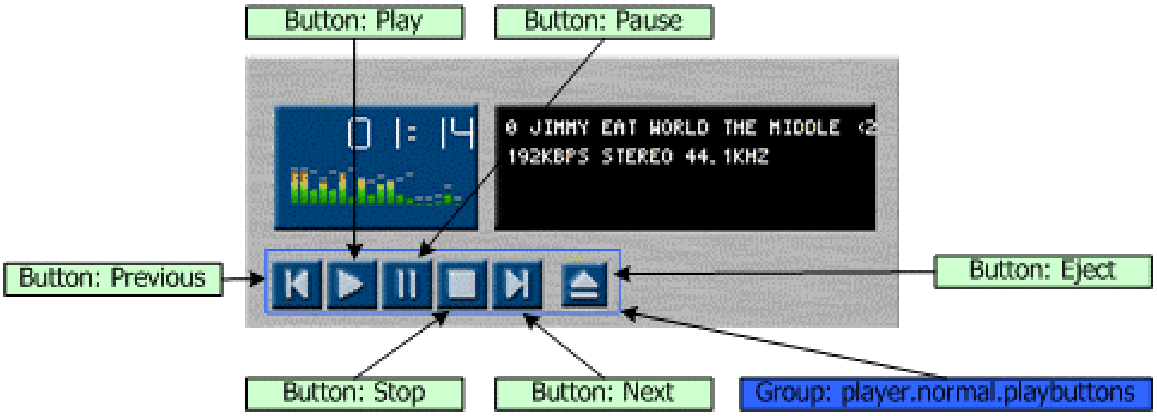
If we change line 113 and 114: change the x&y coordinates of the group objects, their positions are reversed. This is a simple change and much more convenient than changing the position of each element independently.

```
0113 <group id="player.normal" x="0" y="20" />
0114 <group id="player.normal.playbuttons" x="15" y="0" />
```



3. Simple Tutorial: Playback Buttons Group Definition (line 19-64)

```
0019 <!-- Playback Buttons Group Definition -->
0020 <groupdef id="player.normal.playbuttons" name="Player buttons">
0021   <button
0022     id="Previous"
0023     action="PREV"
0024     x="0" y="0"
0025     image="blue-player.button.previous"
0026     tooltip="Previous"
0027   />
0028   <button
0029     id="Play"
0030     action="PLAY"
0031     x="23" y="0"
0032     image="blue-player.button.play"
0033     tooltip="Play"
0034   />
0035   <button
0036     id="Pause"
0037     action="PAUSE"
0038     x="46" y="0"
0039     image="blue-player.button.pause"
0040     tooltip="Pause"
0041   />
0042   <button
0043     id="Stop"
0044     action="STOP"
0045     x="69" y="0"
0046     image="blue-player.button.stop"
0047     tooltip="Stop"
0048   />
0049   <button
0050     id="Next"
0051     action="NEXT"
0052     x="92" y="0"
0053     image="blue-player.button.next"
0054     tooltip="Next"
0055   />
0056   <button
0057     id="Eject"
0058     action="EJECT"
0059     x="121" y="1"
0060     image="blue-player.button.eject"
0061     tooltip="Eject"
0062   />
0063 </groupdef>
0064
```

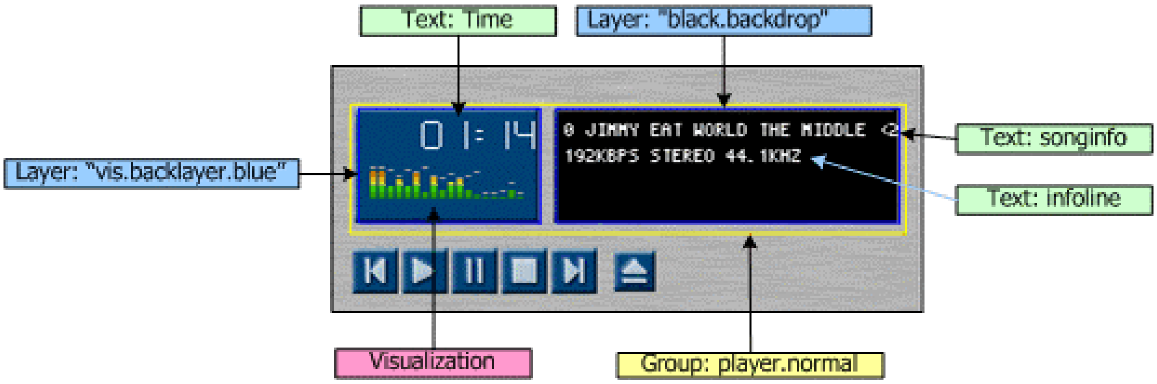


Line 19-64 is a group definition of the playback buttons. This section contains only the definition of the group or, in another word, it defines which objects are in th e group. Once the group is defined, you can use it later as we will see later. Lets go over the elements inside the group.

4. Simple Tutorial: Song File Information and Visualization Group Definition (line 65-107)

```
0065 <!-- Song File Information and Visualization Group Definition -->
0066 <groupdef id="player.normal" name="Winamp">
0067   <layer id="vis.backlayer.blue"
0068    x="13" y="20"
0069    ghost="1"
0070    image="vis.backlayer.blue"
0071   />
0072   <text
0073    id="timer"
0074    display="time"
0075    x="30" y="24"
0076    w="70" h="18"
0077    font="player.BIGNUM" fontsize="21" align="right"
0078    timecolonwidth="7"
0079   />
0080   <layer id="black.backdrop"
0081    x="105" y="20"
0082    ghost="1"
0083    image="black.backdrop"
0084   />
0085   <text
0086    id="songticker" ticker="1"
0087    display="songname" default="" showlen="1"
0088    x="108" y="23"
0089    w="157" h="14"
0090    font="wasabi.font.default" fontsize="13"
0091   />
0092   <!-- Song File Information-->
0093   <text
0094    id="infoline" ticker="1"
0095    display="songinfo" default=""
0096    x="108" y="37"
0097    w="157" h="11"
0098    font="wasabi.font.default" fontsize="12" ghost="1"
0099   />
0100   <!-- Visualzation -->
0101   <vis
0102    id="vis"
0103    x="20" y="46"
0104    w="72" h="16"
0105   />
0106 </groupdef>
0107
```

SongInfo group is another group definition just like the Playback Buttons group. As you can see that the elements defined inside this group is different from the previous one. In this example, we're using elements other than buttons. We're using the layer to set the background color of two areas. A layer is basically an image. Layers can overlay on top of each other. It is similar to a layer in Adobe Photoshop. "Songinfo" and "infoline" are text objects displaying information stored within Winamp3 runtime variables.


Text: Time
Layer: "black.backdrop"
0 1: 14
0 JIMMY EAT WORLD THE MIDDLE
192KBPS STEREO 44.1KHZ
Text: songinfo
Text: infoline
Layer: "vis.backlayer.blue"
Visualization
Group: player.normal

W3 Skinning Tutorial - 2.6 Relative Position

1. What is a relative positioning?
2. RelativePos.xml
3. Relative Position 1: defining X&Y Coordinates
4. Relative Position 2: Specify the Height and Width
5. Relative Position 3: Negative X Coordinate
6. Relative Position 4: Negative Y Coordinate
7. Relative Position 5: Relatw and RelatH

1. What is a relative positioning?

A. Introduction

Built on top of the WASABI platform, Winamp3 skinning offers unparalleled flexibility in element placement. You're no longer restricted to specifying X&Y coordinates for an element. Instead, you can define the position of the element **relative** to your component. When the component resizes, the element will stretch or move with the component. This flexibility frees the skin designer from the hassel of retooling or redrawing the graphic assets over and over again to different size and width. Winamp will do that for you. =)

B. Add relativepos.xml to your SimpleTutorial

For this section, I have made another XML file called **"relativepos.xml"** that you need to include in your SimpleTutorial skin. Download the relativepos.xml here. (Please right-click on the link and choose 'Save As...' or 'Save Link Target As...')
http://www.winamp.com/nsdn/winamp3x/skinning/tutorials/relativepos.xml

Place this file in the same directory as your SimpleTutorial skin directory. For most people, the path is "C:\Program Files\Winamp3\Skins\SimpleTutorial". Add this line to your **Skin.xml** in SimpleTutorial: <include file="relativepos.xml"/>

This is what your skin.xml should look like after the change:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<WinampAbstractionLayer version="0.8">
  <skininfo>
    <version>1.0</version>
    <name>Tutorial Simple Skin</name>
    <comment>Created by pixelpusher for the official Winamp3 skinning site.</comment>
    <author gaybag="be afraid, be very afraid">John Hoag "aka" pixelpusher</author>
    <email>nonsaint@hotmail.com</email>
    <homepage></homepage>
    <screenshot>screenshot.png</screenshot>
  </skininfo>
  <include file="@DEFAULTSKINPATH@xml/eq.xml"/>
  <include file="@DEFAULTSKINPATH@xml/thinger.xml"/>
  <include file="@DEFAULTSKINPATH@xml/pledit.xml"/>
  <include file="@DEFAULTSKINPATH@xml/player-elements.xml"/>
  <include file="@DEFAULTSKINPATH@xml/pledit-normal-group.xml"/>
  <include file="@DEFAULTSKINPATH@xml/video.xml"/>
  <include file="xml/player.xml"/>
  <include file="relativepos.xml"/>   <-- Add this line
</WinampAbstractionLayer>
```

2. RelativePos.xml

LINE NUMBER CODE

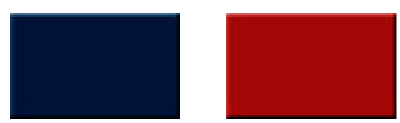
```
<!-- Simple Tutorial Skin - Relativepos.xml file  -->
<!-- Advanced topic: Relative position.  -->
<!-- Created by Ken Chen for the official Winamp3 skinning site. -->

```

Relativepos.xml uses elements from the SimpleTutorial skin. I am using the "blue -vis-layer.png" and "red-vis-layer.png" in the Player directory.

```
<!-- Element definition -->
<elements>
  <bitmap id="blue" file="player/blue-vis-layer.png" gammagroup="Backgrounds"/>
  <bitmap id="red" file="player/red-vis-layer.png" gammagroup="Backgrounds"/>
</elements>

```



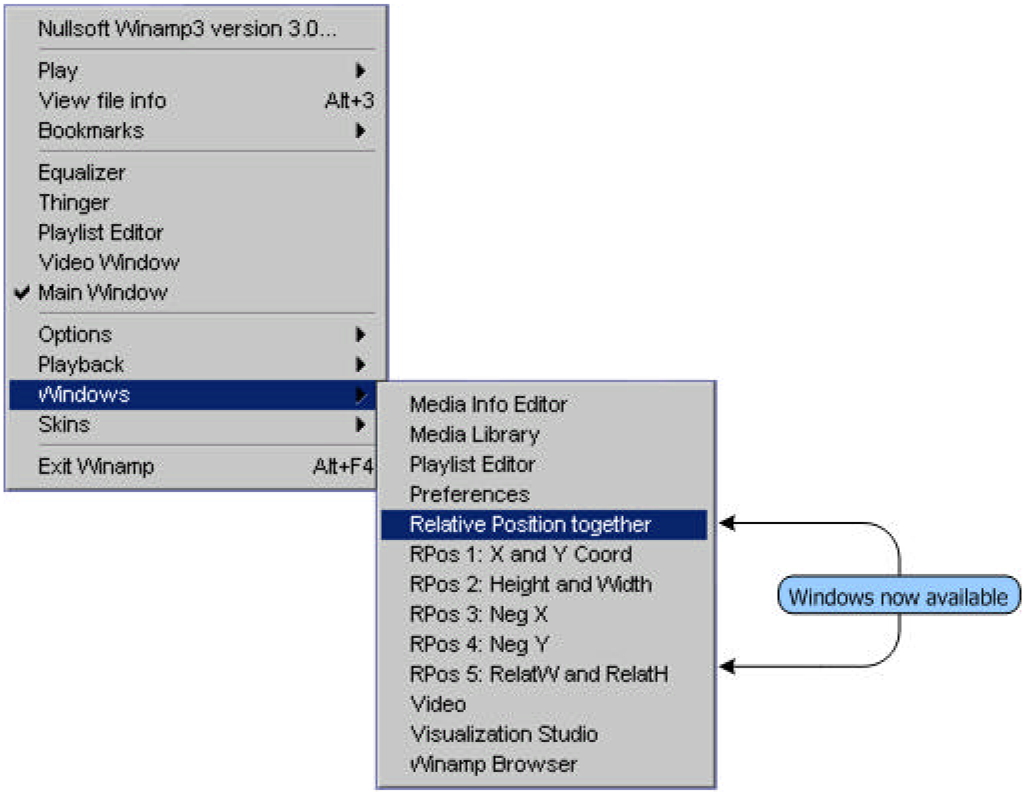
3. Relative Position 1: defining X&Y Coordinates

```
<!-- Group Definition 1: X&Y Coordinate; -->
<groupdef id="relativepos1" register_autopopup="1" name="RPos 1: X and Y Coord">
  <layer id="vis.blue"
    x="20" y="30"
    ghost="1"
    image="blue"
  />
</groupdef>

```

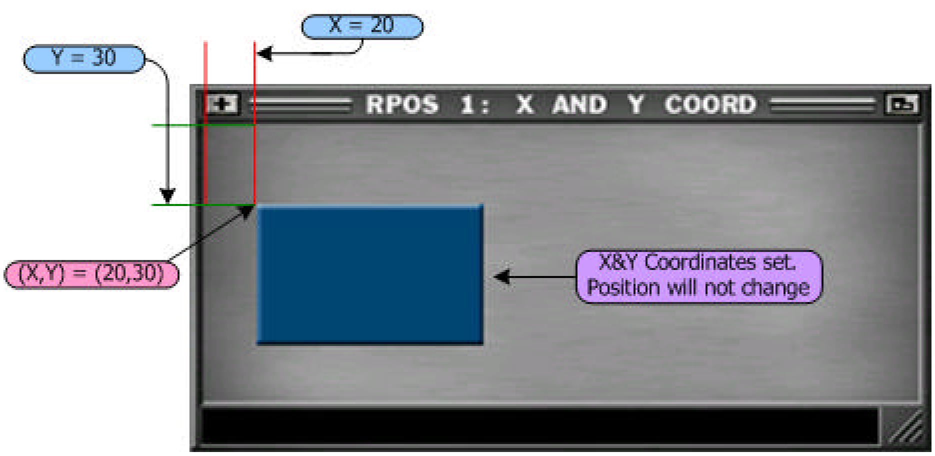
A. register_autopopup="1"

Before we delve into the code, I want to go over one feature. Take a look at this tag: `register_autopopup="1"` When you add this tag to a groupdef, Winamp3 automatically wrap a container and layout around it and allow you to see the group in a separat e window. This is particularly useful when you want to test the behavior of that group separate from everything else. To open that window, right click on the Main Player to bring up the menu. Go to the "Windows" submenu and you'll see the extra windows rel ativepos.xml has created. Open them up and try them out.

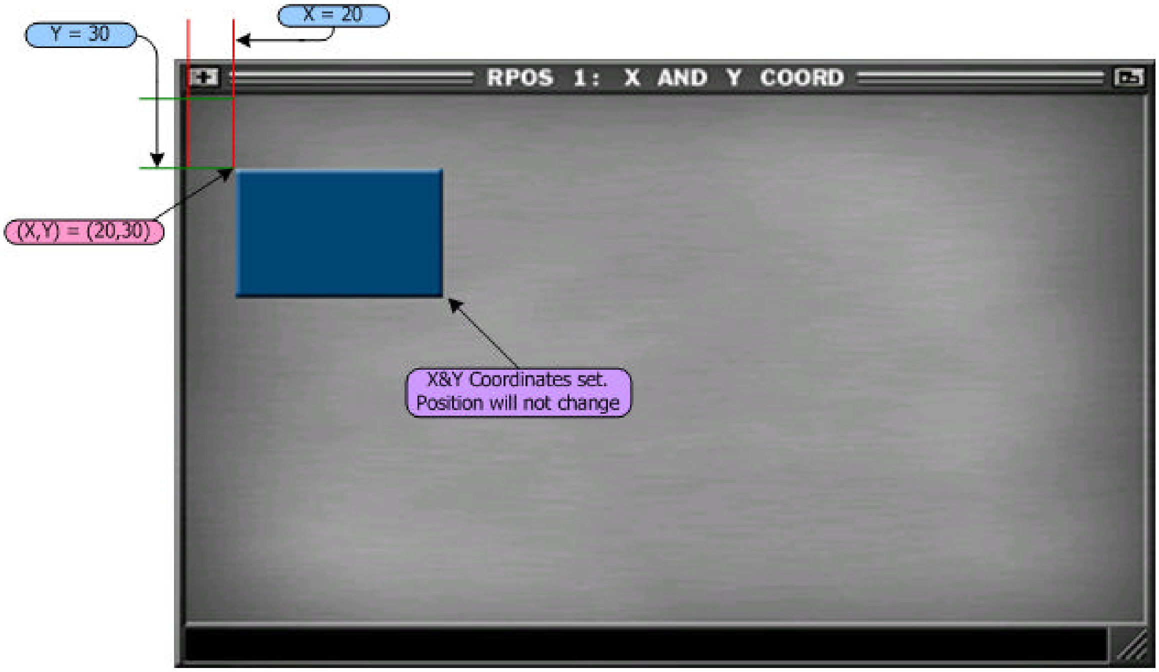


B. Defining X&Y Coordinates

Line 11-19 defines a group with a "blue -vis" layer at the coordinate of (20,30). This is probably what you've been doing all along. As you can see, that the blue -vis layer's location doesn't change if you resize the window. The position is fixed.



***Small size window**



***Large size window; Blue-vis layer position did not change**

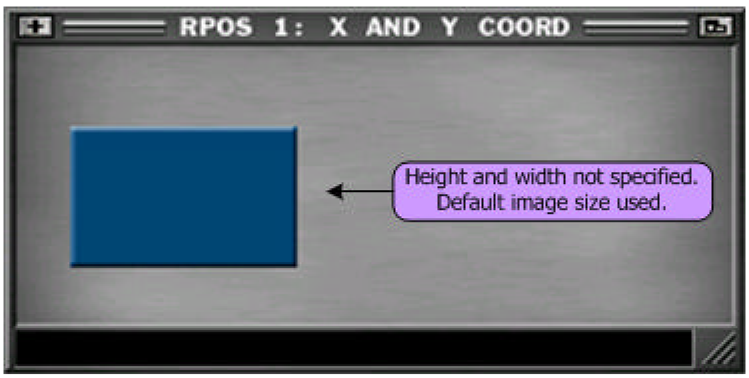
I encourage you to play with this XML file. If you change the X&Y coordinates on line 14 to (-20, -30), your blue-vis layer will disappear. The reason is that the coordinate specified will be outside the context of your component window and therefore will not be rendered.

***Negative X&Y Coordinates: (-20, -30)**

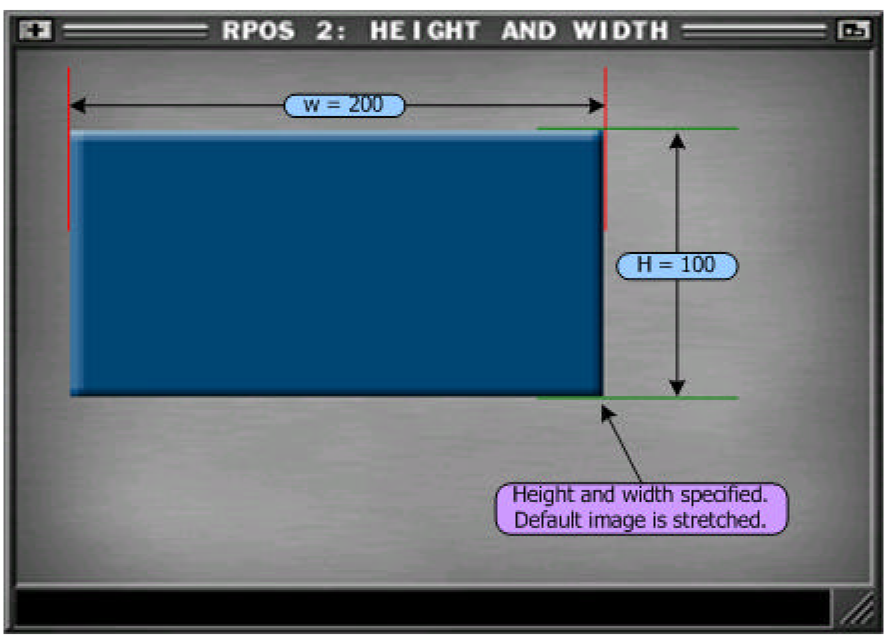
4. Relative Position 2: Specify the Height and Width

```
<!-- Group Definition 2: Specify the Height and Width -->
<groupdef id="relativepos2" register_autopopup="1" name="RPos 2: Height and Width">
  <layer id="vis.blue"
    x="20" y="30"
    h="100" w="200"
    ghost="1"
    image="blue"
  />
</groupdef>
```

In group 2, we're defining the height and width of the blus -vis layer that is larger than the image. When you do that, Winamp will stretch the image to that size. Likewise, if you specify height or width that is smaller than the image, Winamp will shrink the image to fit to that size.



***RPOS1 window: Height and Width not specified**



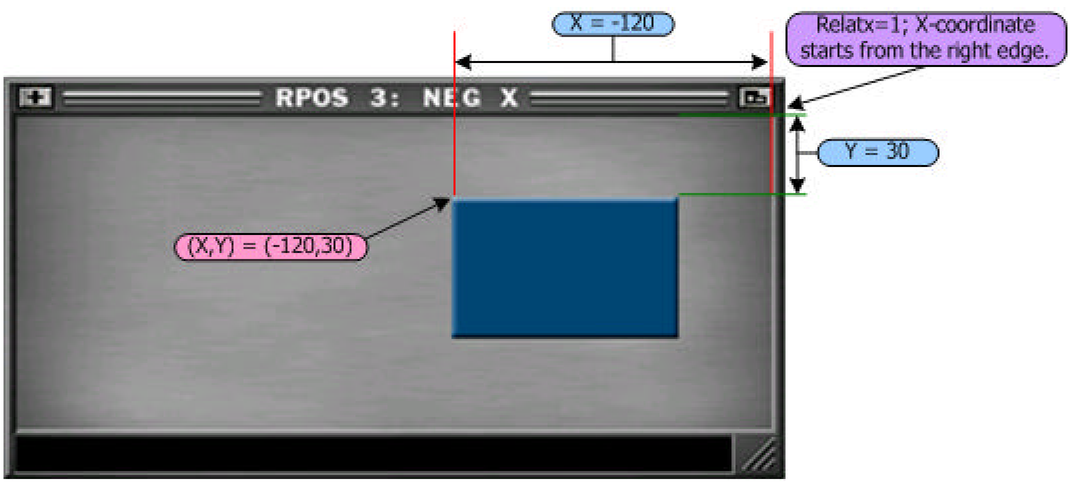
***RPOS2 window: Height and Width defined**

5. Relative Position 3: RelatX & Negative X Coordinate

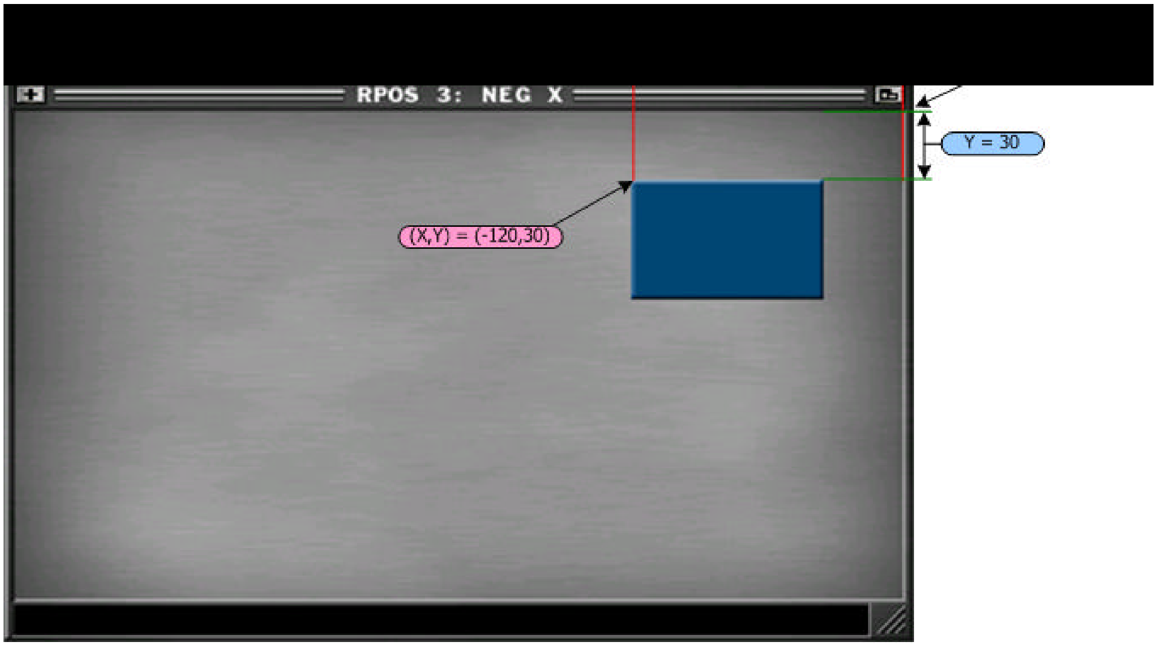
```
0030 <!-- Group Definition 3: RelatX & Negative X Coordinate -->
0031 <groupdef id="relativepos3" register_autopopup="1" name="RPos 3: Neg X">
0032   <layer id="vis.blue"
0033     x="-120" y="30"
0034     relatx="1"
0035     ghost="1"
0036     image="blue"
0037   />
0038 </groupdef>
0039
```

Line 30-39 defines a group with a "blue-vis" layer at the coordinate of (-120,30). Notice that we have an extra line on line 34: "relatx=1". By setting relatx to 1, you're telling Winamp to start the x - coordinate from the right edge of the window. Note that the X -Coordinate should be negative; otherwise the blue-vis layer will not show up.

This is a convenient feature if you want an element stick to a specific distance relative to the right edge of the window. Notice that when I resize the window, the blue -vis layer stays fixed distance relative to the right edge of the window.



***RPOS3 window with relatx=1**



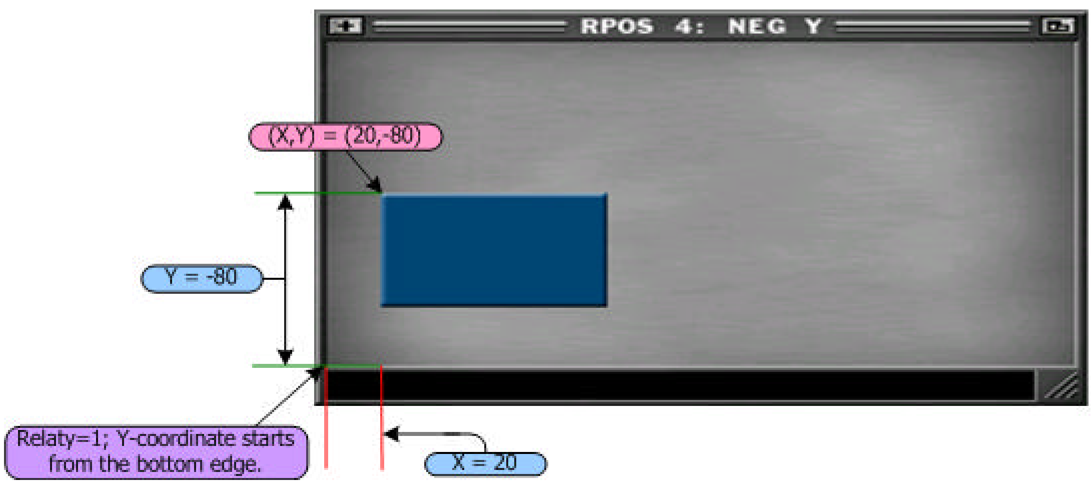
***RPOS3 window with relatx=1: coordinate (20,30)**

6. Relative Position 4: RelatY & Negative Y Coordinate

```
0040 <!-- Group Definition 4: RelatY & Negative Y Coordinate -->
0041 <groupdef id="relativepos4" register_autopopup="1" name="RPos 4: Neg Y">
0042   <layer id="vis.blue"
0043     x="20" y="-80"
0044     relaty="1"
0045     ghost="1"
0046     image="blue"
0047   />
0048 </groupdef>
0049
```

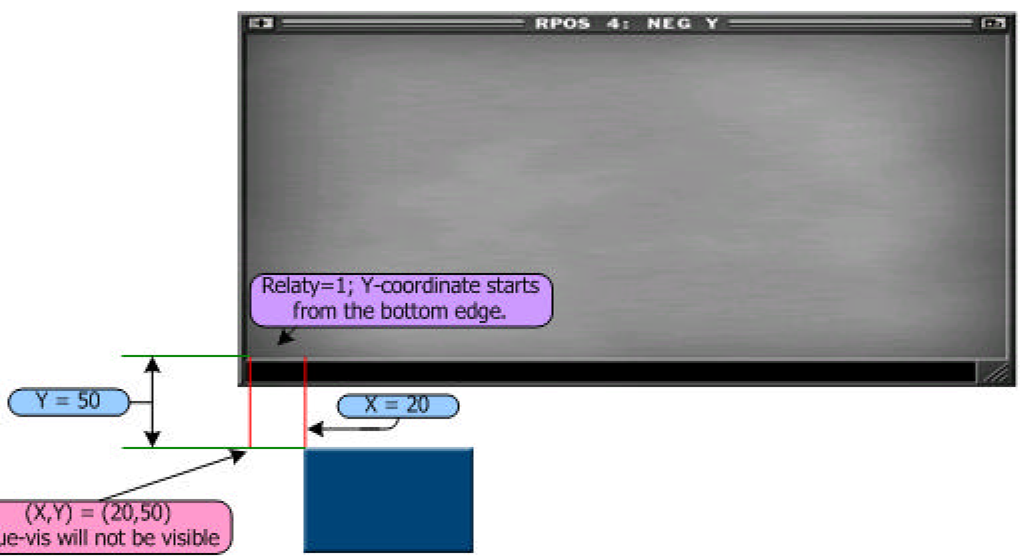
Line 40-49 defines a group with a "blue -vis" layer at the coordinate of (20, -80). Notice theline on line 44: "relaty=1". By setting relaty to 1, you're telling Winamp to start the y -coordinate from the bottom edge of the window. Note that the Y -Coordinate should be negative; otherwise the blue-vis layer will not show up.

This is a convenient feature if you want an element stick to a specific distance relative to the bottom edge of the window. Notice that when I resize the window, the blue -vis layer stays fixed distance relative to the bottom edge of the window.



*RPOS4 window with relaty=1

***RPOS4 window with relaty=1: window resized**



***RPOS4 window with relaty=1: coordinate (20,50)**

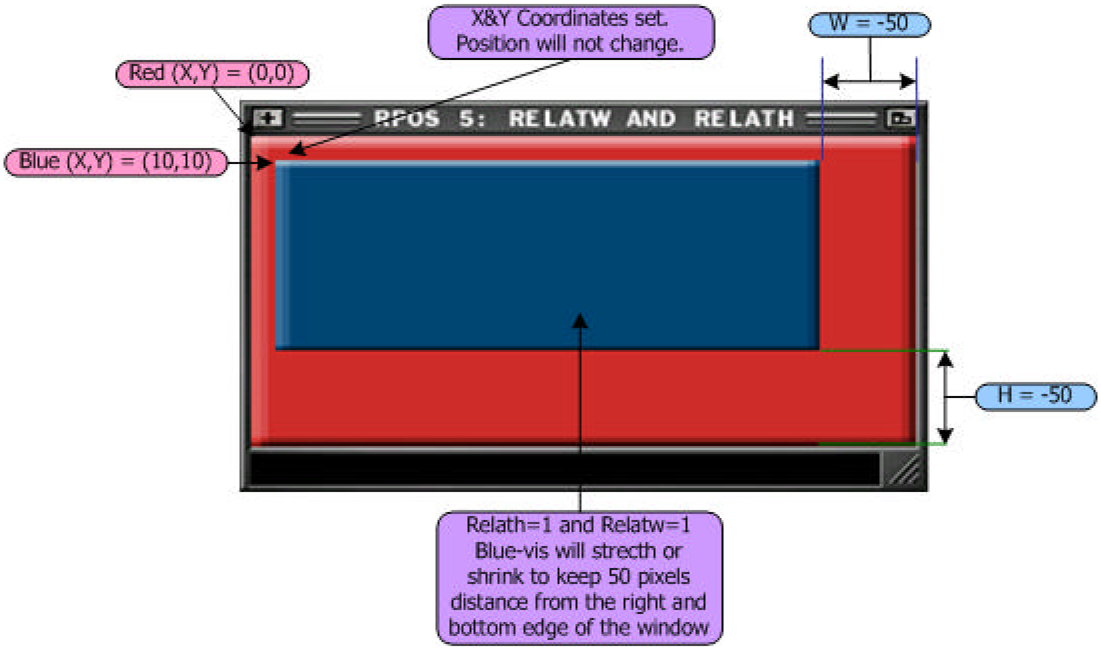
7. Relative Position 5: Relatw and RelatH

```
0050 <!-- Group Definition 5: Relatw and RelatH -->
0051 <!-- a. Group Definition: inside blue squre-->
0052 <groupdef id="relativepos.blue" name="RPos: blue">
0053   <layer id="vis.blue"
0054     x="0" y="0"
0055     h="0" w="0"
0056     relath="1"     relatw="1"
0057     ghost="1"
0058     image="blue"
0059   />
0060 </groupdef>
0061
0062 <!-- b. Group Definition: Outside red squre containing blue square-->
0063  <groupdef id="relativepos.red" register_autopopup="1" name="RPos 5: RelatW and RelatH">
0064   <layer id="vis.red"
0065     x="0" y="0"
0066     h="0" w="0"
0067     relath="1"     relatw="1"
0068     ghost="1"
0069     image="red"
0070   />
0071   <group id="relativepos.blue"  x="10" y="10" w="-50" h="-50" relatw="1" relath="1"/>
0072 </groupdef>
```

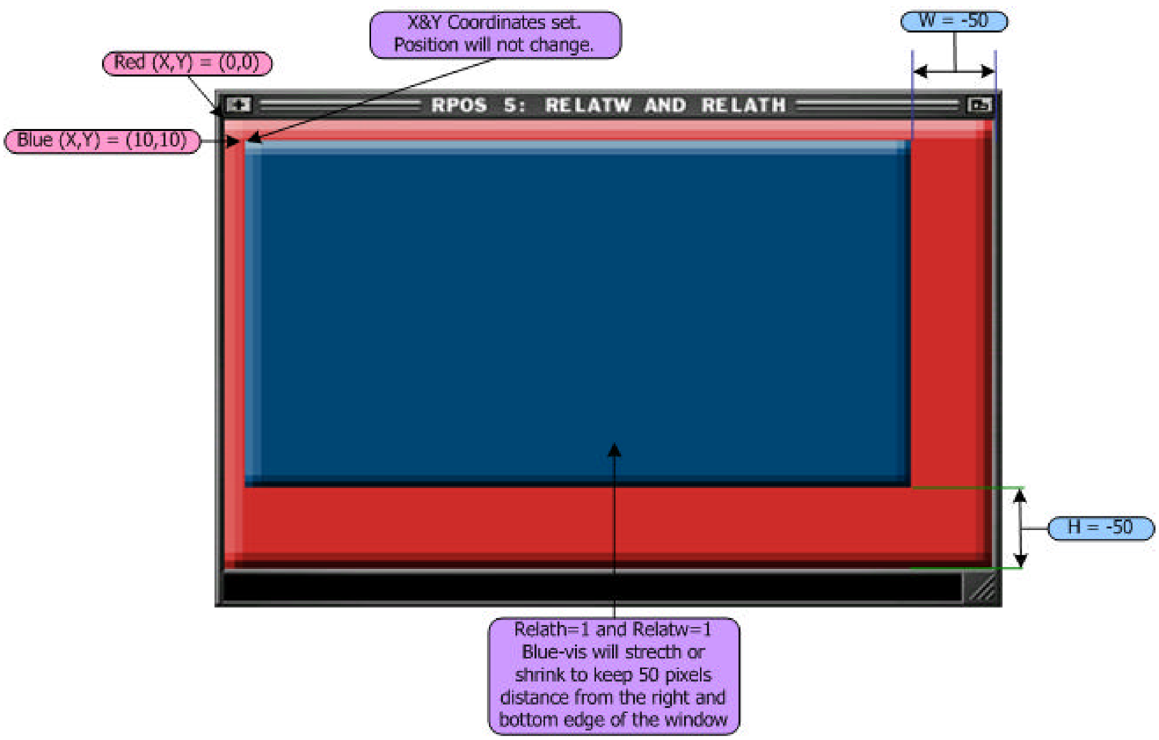
Line 50-72 demonstrate the use of relath and relatw. From the beginning of this section we know that by defining height and width different from the original image size, Winamp3 will automatically stretch or shrink the image to fit that size constraint. What is powerful (and awesome) about Winamp3 is that you can also tell Winamp3 to adjust the height and width of an element with respect to the right and bottom edge of the window. This is accomplished through "relath" and "relatw".

In our example, we defined a blue-vis group (lines 52-60) and a red-vis group that encompasses the blue-vis group (lines 63-72). The blue-vis group is defined with no preset x, y, h or w because we will define them on line 71.

The magic happens on line 71. With relath set to 1 and relatw set to 1, we're telling Winamp3 to stretch or shrink the blue-vis layer so that it is 50 pixels away from the right and bottom edge of the window. The blue-vis layer is stretched and the distance is maintained as I expanded the size of the window.

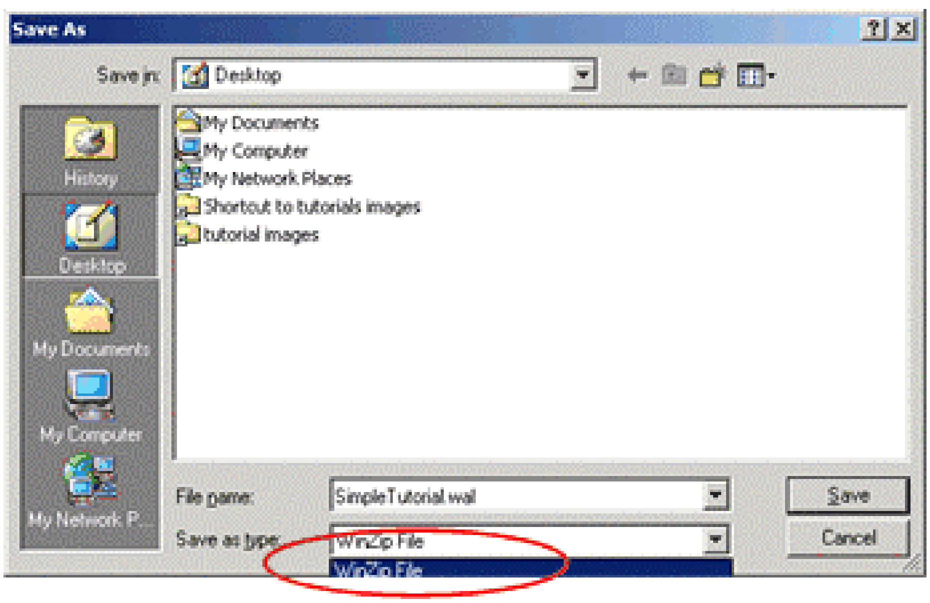


*RPOS5 window with relath=1 and relatw=1



*RPOS5 window with relath=1 and relatw=1; window resized; Blue-vis layer stretched

- When you save the file, make sure that the type of the file is selected to "All Type" and **NOT** "Winzip" filetype. You should save the file into your skin subdirectory in your Winamp3 directory. For most people, that path is "C:\Program Files\Winamp3\Skins".



Contact Us

Please email Ken Chen if you have any suggestion.

W3 Skinning Tutorial - 3.1 Non-Rectangular Players

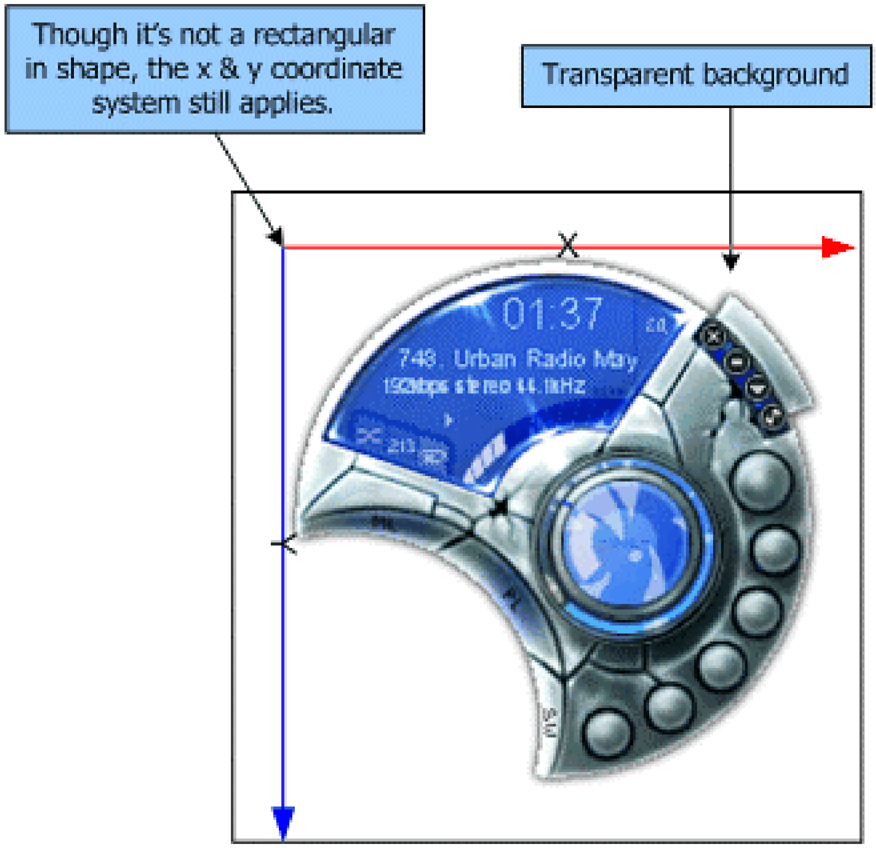
1. Introduction
2. Complex Tutorial Skin - The Break Down
3. Player.XML - Part 1: SimpleTutorialXML.xml
4. Player.XML - Part 2: Graphic Elements

1. Introduction

Winamp3 brings forth unprecedented flexibility and power to skinning. You are no longer confined to simple rectangular shapes; in fact any shape is possible. Just take a look at these awesome examples. In addition, Winamp3 has been revamped so that any part of the user interface is skinnable.

You should notice that while the shapes are different, the X and Y coordinate system still applies. The (0,0) coordinate starts at the upper-left corner. The x-coordinate increases at you move to the right and the y-coordinate increases as you move down.

While the shape of the image can be anything you want, the image itself remains in rectangular form. What happens is that the background of the image is transparent; therefore all you see is the shape that you intend to show. To produce the complex designs that you see below, you must overlay many images on top of the each other. To do this, you must use layers. Layer composition is covered in the next section.



2. Complex Tutorial Skin - The Break Down

Complex Tutorial Skin is composed of these parts:

- Everything from Simple Tutorial Skin
- A second container: "complex"
- Within the "complex" container, a layout called "normal".

  - Group #1: "complex.circle" - to illustrate Layer Composition (Section 3.2)
  - Group #2: "complex.circle2" - to illustrate Alpha Channel (Section 3.3)
  - Group #3: "complex.circle3" - to illustrate Drawer (Section 3.6 and 4.1)
  - Group #4: "complex.circle4" - to illustrate Animated Layer (Section 3.4)
  - Group #5: "complex.circle5" - to illustrate Animating Skin (Section 4.2)

Concepts such as container, layout and group were explained in Section 2. Please refer to them.

```xml
<!-- Element definition -->
<elements>
  <bitmapfont id="player.BIGNUM" file="player/numfont.png" charwidth="9"
   charheight="13" hspacing="3" vspacing="1" gammagroup="Fonts"/>
  <bitmapfont id="wasabi.font.default" file="player/default-font.png" charwidth="5"
   charheight="6" hspacing="0" vspacing="0" gammagroup="Display Fonts"/>
  <bitmap id="player.main.background" file="player/background.png"
   gammagroup="Backgrounds"/>
  <bitmap id="complex.redcircle" file="player/red-circle.png" gammagroup="Circles"/>
  <bitmap id="complex.yellowcircle-arrow" file="player/yellowcircle-arrow.png"
   gammagroup="Circles"/>
  <bitmap id="complex.triangle" file="player/triangle.png" gammagroup="Triangle"/>
  <bitmap id="complex.triangle.clickme" file="player/triangle-clickme.png"
   gammagroup="Triangle"/>
</elements>

```

W3 Skinning Tutorial - 3.2 Layer Composition

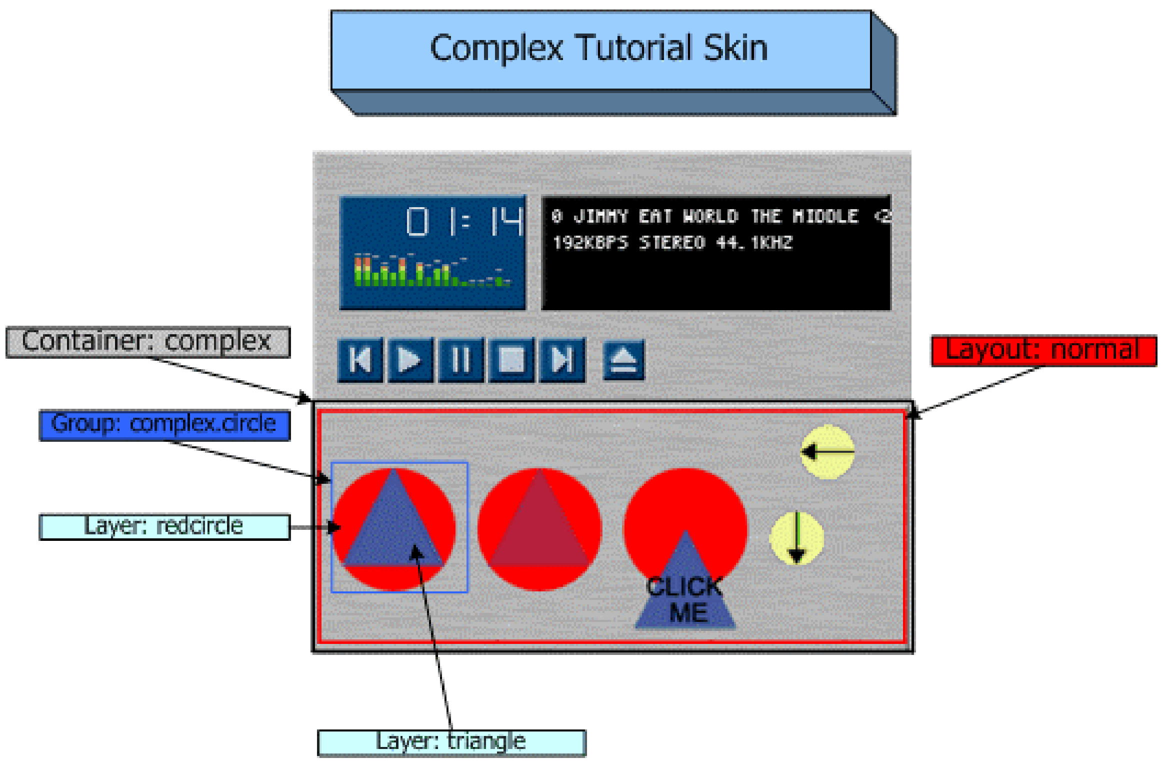
1. Layer Composition

Layer composition is the process by which several images lay on top of each other to produce a complex graphical visualization. It is similar to layers in Photoshop.

The process is:

1) produce the images and save them in a file
2) define an element to represent that image in your XML
3) make a layer using that element
4) if you put layers in sam e area, they will lay on top of each other based on the ordering of the xml code.

2. Complex Tutorial Skin



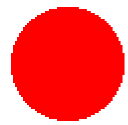
In Complex Tutorial, Group #1 (the first circle from the left), has blue triangle on top of the red circle. This is done through layer composi tion. Here's the process.

a) Make the graphics: the red circle and blue triangle.
b) Define the elements in the XML code.
c) Define layers using those elements.
d) Place the layers in the same area.

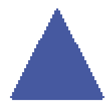
Lets go through in detail how do to each step:

a) make the graphics: the red circle and blue triangle.

red-circle.png



triangle.png



b) define the elements in the XML code.

```
0013 <bitmap id="complex.redcircle" file="player/red-circle.png" gammagroup="Circles"/>
0014 ......
0015 <bitmap id="complex.triangle" file="player/triangle.png" gammagroup="Triangle"/>
```

c) define layers using those elements.

```
0019 <!-- Circle #1 Group Definition: Layer Composition -->
0020 <groupdef id="complex.circle" name="Red Circle">
0021   <layer id="redcircle"
0022     x="10" y="30"
0023     ghost="0"
0024     image="complex.redcircle"
0025   />
```

Group #1: Red circle definition

1st Layer: red circle
Location of the layer
ghost = 0 visible; 1 invisible
Using "complex.redcircle" image element defined on line 13

```
0026   <layer id="triangle"
0027     x="10" y="30"
0028     ghost="0"
0029     image="complex.triangle"
0030   />
0031 </groupdef>
```

2nd Layer: blue triangle
Location of the layer
ghost = 0 visible; 1 invisible
Using "complex.triangle" image element defined on line 15

d) place the layers in the same area.

```
0019 <!-- Circle #1 Group Definition: Layer Composition -->
0020 <groupdef id="complex.circle" name="Red Circle">
0021   <layer id="redcircle"
0022     x="10" y="30"
0023     ghost="0"
0024     image="complex.redcircle"
0025   />
0026   <layer id="triangle"
0027     x="10" y="30"
0028     ghost="0"
0029     image="complex.triangle"
0030   />
0031 </groupdef>
```

- Same X & Y coordinate.
- Red circle and blue triangle have the height and width
- Triangle code comes after redcirlce => Triangle image is on top of circle

W3 Skinning Tutorial - 3.3 Alpha Channels

1. Alpha Channel

Alpha channel controls the transparency value of an element. By adjusting the alpha value of an element, you can make the element totally transparent or not transparent at all. Winamp3 blends that element with the background image if the transparency of that element is increased.

Take a look at the two images below. Both images have a red rectangle layer on top of the Boxer skin. The red rectangle in the first image has no transparency: the result is an overlap of images. The second image uses alpha channel to create some transparency: the result is the blending of the red rectangle and the Boxer skin.

- No transparency: images overlap.

* Some transparency: images blend.

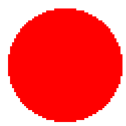
2. Complex Tutorial Skin

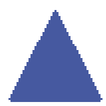
In Complex Tutorial, Group #2 (the second circle from the left), has blue triangle blended with the red circle. This is done through Alpha Channel. Note that the graphical asset for this group is the same as the ones used to demonstrate Layer Composition. The only difference is that we adjusted the alpha value for the triangle in this section. Here's the process.

a) Make the graphics: the red circle and blue triangle.
b) Define the elements in the XML code.
c) Define layers using those elements.
d) Place the layers in the same area.
e) Change the alpha value for the triangle.

Lets go through in detail how do to each step:

a) make the graphics: the red circle and blue triangle.

red-circle.png 

triangle.png 

b) define the elements in the XML code.

```
0013 <bitmap id="complex.redcircle" file="player/red-circle.png" gammagroup="Circles"/>
0014 ......
0015 <bitmap id="complex.triangle" file="player/triangle.png" gammagroup="Triangle"/>
```

c) define layers using those elements.

Code	Explanation
0032	
0033 `<!-- Circle #2 Group Definition: Alpha Channel -->`	<u>Group #2: Red circle definition</u>
0034 `<groupdef id="complex.circle2" name="Red Circle 2">`	
0035 `<layer id="redcircle2"`	1st Layer: red circle
0036 `x="77" y="30"`	Location of the layer
0037 `ghost="0"`	ghost = 0 visible; 1 invisible
0038 `image="complex.redcircle"`	Using "complex.redcircle" image
0039 `/>`	element defined on line 13

```
0040   <layer id="triangle2"
0041    x="77" y="30"
0042    ghost="0"
0043    image="complex.triangle"

0044    alpha="100"
0045   />            <!-- alpha between 0 and 255 -->
0046  </groupdef>
```

2nd Layer: blue triangle
Location of the layer
ghost = 0 visible; 1 invisible
Using "complex.triangle" image element defined on line 15

d) place the layers in the same area.

```
0032
0033  <!-- Circle #2 Group Definition: Alpha Channel -->
0034  <groupdef id="complex.circle2" name="Red Circle 2">
0035   <layer id="redcircle2"
0036    x="77" y="30"
0037    ghost="0"
0038    image="complex.redcircle"
0039   />
0040   <layer id="triangle2"
0041    x="77" y="30"
0042    ghost="0"
0043    image="complex.triangle"
0044    alpha="100"
0045   />            <!-- alpha between 0 and 255 -->
0046  </groupdef>
```

- Same X & Y coordinate.
- Red circle and blue triangle have the height and width
- Triangle code comes after redcirlce => Triangle image is on top of circle

e) Change the alpha value for the triangle.

```
0033  <!-- Circle #2 Group Definition: Alpha Channel -->
......  .......
0040   <layer id="triangle2"
0041    x="77" y="30"
0042    ghost="0"
0043    image="complex.triangle"
0044    alpha="100"
0045   />            <!-- alpha between 0 and 255 -->
0046  </groupdef>
```

- Alpha value specifies the transparency of an element.
- 0 is totally transparent
- 255 is no transarency

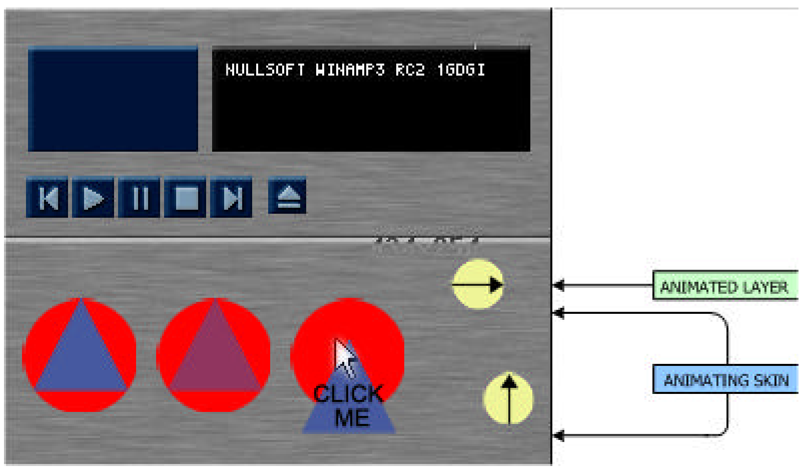
W3 Skinning Tutorial - 3.4 Animated Layer

1. Animated Layer
2. Complex Tutorial Skin

1. Animated Layer

An animated layer is a layer that displays a series of graphics by cycling through parts of a graphics file. A simple example is the rotating arrow that you see below.

An animated layer does not move. However, an animating skin does move. To move a graphical element in animating layer, you must write a MAKI script. Scripting is covered in the next section.



2. Complex Tutorial Skin

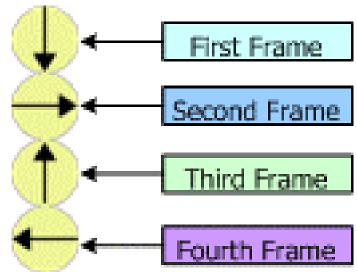
In Complex Tutorial, Group #4 shows a spinning arrow inside a yellow circle. This is done through Animated Layer. Here's the process.

a) Make the graphics: rotating arrow
b) Define the elements in the XML code.
c) Define layers using those elements.

Lets go through in detail how do to each step:

a) Make the graphics: the red circle and blue triangle.

When you make the graphics for this layer, make the graphics for each frame separately. Then put the frames together vertically in a single file as shown below. This example has a short animation. If you plan to show more complicated animation, make more frames.



b) define the elements in the XML code.

The definition for the element is the same as any bitmap definition. The thing that is different is the next step.

```
0032
0033 <!-- Circle #2 Group Definition: Alpha Channel -->
0034 <groupdef id="complex.circle2" name="Red Circle 2">
0035   <layer id="redcircle2"
0036     x="77" y="30"
0037     ghost="0"
0038     image="complex.redcircle"
0039   />
```

Group #2: Red circle definition

1st Layer: red circle
Location of the layer
ghost = 0 visible; 1 invisible
Using "complex.redcircle" image element defined on line 13

c) Define layers using those elements.

Take a look at line 70 . The width and height tells Winamp3 how big each frame is. It will cycle through each frame automatically.

```
0040   <layer id="triangle2"
0041     x="77" y="30"
0042     ghost="0"
0043     image="complex.triangle"

0044     alpha="100"
0045   />              <!-- alpha between 0 and 255 -->
0046 </groupdef>
```

2nd Layer: blue triangle
Location of the layer
ghost = 0 visible; 1 invisible
Using "complex.triangle" image element defined on line 15

W3 Skinning Tutorial - 3.5 Snap Points

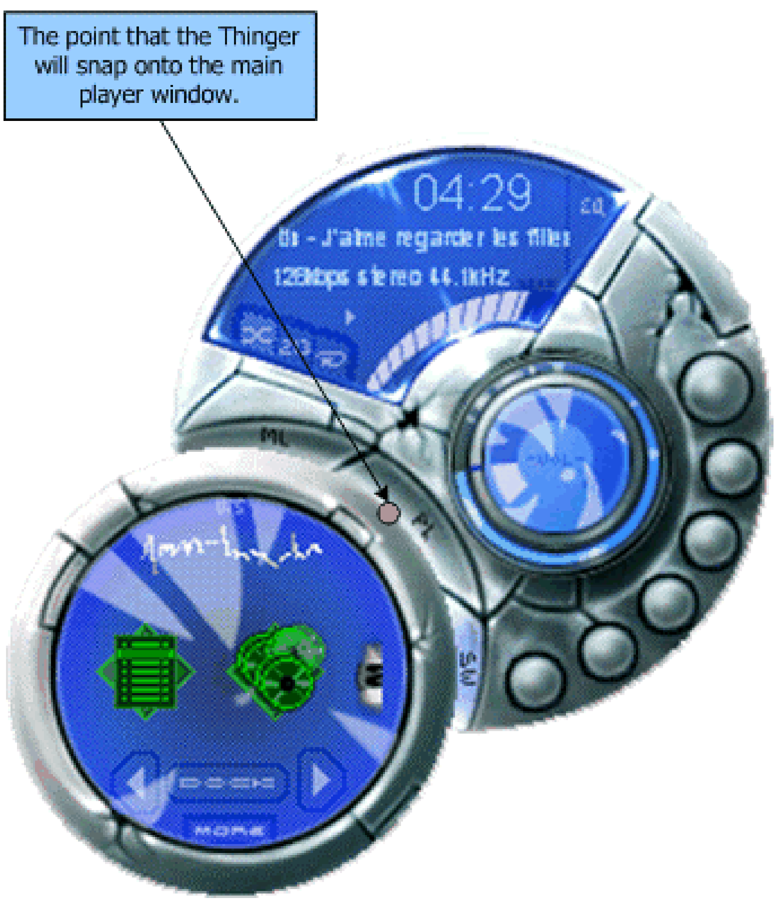
1. Snap Points
2. Boxor Example
3. Default Player Example
4. XML example

1. Snap Points

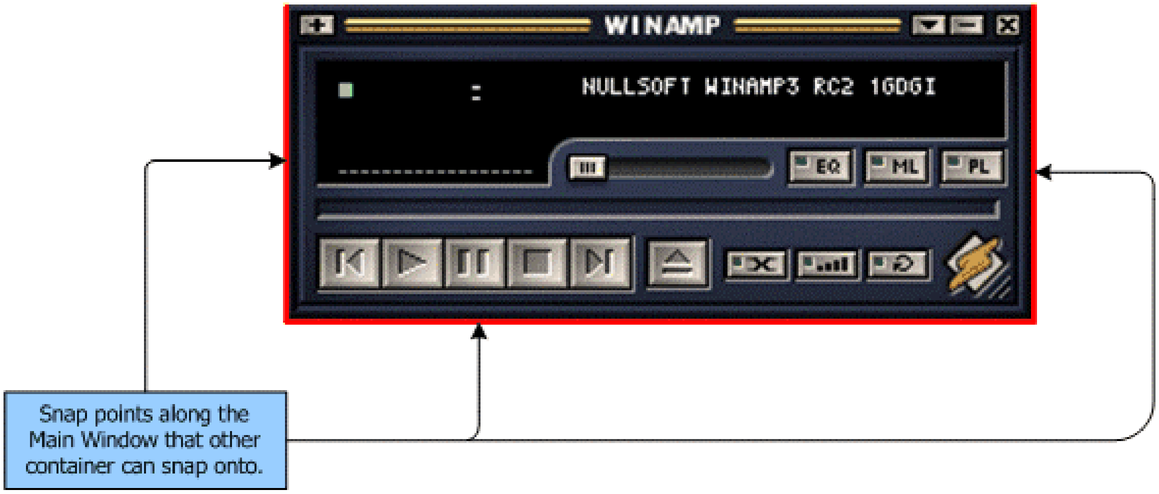
Snap point is a point that windows will snap on to. When you move that window close in proximity to that point, the window will "latch" on to that point. You can drag it out of that point later. Take a look at these two examples.

2. Boxor Example

Snap point is a useful tool that adds more functionality to your skin. Take a look at the Boxor example.



3. Default Player Example



- Snap points defined along the Main Window

- The Playlist Editor snaps to the Main Window when you move it close in proxim ity of the Main Window

4. XML Example

Lets take a look at the XML code to define a snappoint. To get two components to snap together when they're close, you need to define snappoints in their XML. The XML itself is simple. The important part is the id: components with the same snappoint id will snap together whereas components with different ids will not.

```
<!-- file: FirstComponent.XML -->
<!-- Snappoint defined in the first component -->
.........
<Snappoint
  id="round"
  x="97"
  y="151"
/>
```

Snappoints

- id: "round" matches with the snappoint defined in the second component.
- Location of the snappoint

```
<!-- file: SecondComponent.XML -->
<!-- Snappoint defined in the second component -->
........
<Snappoint
  id="round"
  x="127"
  y="21"
/>
```

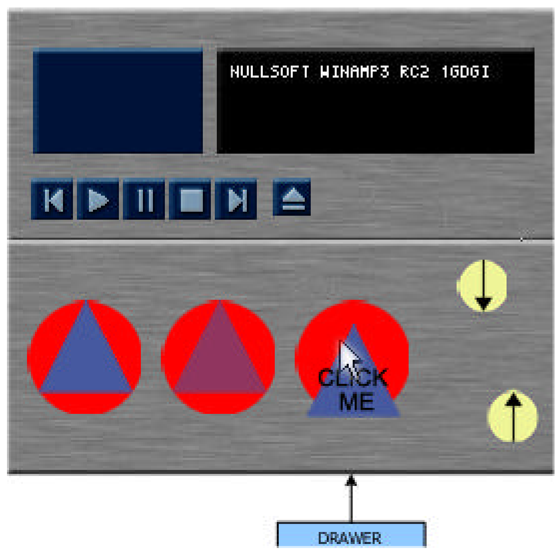
Snappoints

- id: "round" matches with the snappoint defined in the first component.
- Location of the snappoint

W3 Skinning Tutorial - 3.6 Drawers

1. Drawers

A Drawer is an animated layer that is normally hidden but can be shown when triggered by a user event (often a mouse click).



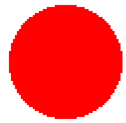
2. Complex Tutorial Skin

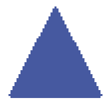
In Complex Tutorial, Group #3 shows a triangle that slides out when you click on it. This group demonstrates the concept of a drawer. We will go over the XML code in this section. The script explanation is covered in the next section. Here's the process.

a) Make the graphics: rotating arrow
b) Define the elements in the XML code.
c) Define layers using those elements.
d) Making MAKI script.

Lets go through in detail how do to each step:

a) make the graphics: the red circle and blue triangle.

red-circle.png 

triangle.png 

b) define the elements in the XML code.

```
0013 <bitmap id="complex.redcircle" file="player/red-circle.png"
      gammagroup="Circles"/>
......
0016 <bitmap id="complex.triangle.clickme" file="player/triangle-clickme.png"
      gammagroup="Triangle"/>
```

c) define layers using those elements.

```
0048 <!-- Circle #3 Group Definition: Drawer -->
0049 <groupdef id="complex.circle3" name="Red Circle 3">
0050   <layer id="redcircle3"
0051     x="144" y="30"
0052     ghost="0"
0053     image="complex.redcircle"
0054   />
0055   <script id="complex" file="scripts/triangle.maki"/>
0056   <button id="complex.circle3.triangle3"
0057     x="144" y="30"
0058     ghost="0"
0059     image="complex.triangle.clickme"

0060     move="0"
0061     rectrgn="0"
0062     sysregion="1"
0063   />          <!-- Y: 30 closed, 60 open -->
0064 </groupdef>
```

Group #3: Drawer Group

- Layer: "redcircle3"
- Location of the layer
- Ghost = 0 visible; 1 invisible
- Using "complex.redcircle" image (line 13)
- Using scripting
- Button: "complex.circle3.triangle3"
- Location of the button
- Ghost = 0 visible; 1 invisible
- Using "complex.redcircle" image (line 13)
- Is this a moving layer: No
- Rectangular shape: No
- Extend to desktop rendering

Options: (0=No, 1=Yes)

d) Making MAKI script

Two MAKI scripts function together to create the animations you see. Triangle.maki animates the sliding effect of the drawer. Anim.maki animates the moving arrow. We will go over the MAKI script for drawer in the next section. Read on...

```
0001 // Complex Tutorial Skin - triangle.m MAKI file
0002 // Created by Ken Chen for the official Winamp3 skinning site.
0003
```

```
//  #include Section
#include "../../../lib/std.mi"
#include "../../../lib/config.mi"
#include "../../../lib/pldir.mi"

//  Variable Declaration Section
global Button button_triangle;
global Int moving_triangle;
global String status_triangle;

//  System.onScriptUnloading() Section
System.onScriptUnloading(){
}

//  System.onScriptLoaded() Section
System.onScriptLoaded(){
  group triangleDrawer = getScriptGroup();
  button_triangle = triangleDrawer.findobject("complex.circle3.triangle3");
  status_triangle = "closed";
  moving_triangle = 0;
}

//  Your own functions
button_triangle.onleftbuttonup(int x,int y){
  if( status_triangle == "closed" && moving_triangle != 1 ){
                moving_triangle = 1;
                button_triangle.setTargetY(60);
                button_triangle.setTargetSpeed(0.25);
                button_triangle.gototarget();
  }else if( status_triangle == "open" && moving_triangle != 1){
                moving_triangle = 1;
                button_triangle.setTargetY(30);
                button_triangle.setTargetSpeed(0.15);
                button_triangle.gototarget();
  }
}

button_triangle.ontargetreached(){
  if( status_triangle == "closed" ){
                status_triangle = "open";
                moving_triangle = 0;
  }else if( status_triangle == "open" ){
                status_triangle = "closed";
                moving_triangle = 0;
  }
}
```

- Triangle.m MAKI file for Complex Tutorial Skin

```
//  Complex Tutorial Skin - anim.m MAKI file
//  Created by Ken Chen for the official Winamp3 skinning site.

//  #include Section
#include "../../../lib/std.mi"
#include "../../../lib/config.mi"
#include "../../../lib/pldir.mi"

//  Variable Declaration Section
function move_arrow();

global String status_arrow;
global AnimatedLayer arrow;

//  System.onScriptUnloading() Section
System.onScriptUnloading(){
}

//  System.onScriptLoaded() Section
System.onScriptLoaded(){
		group arrowAnim = getScriptGroup();
		arrow = arrowAnim.findobject("arrow");
		status_arrow = "down";
		move_arrow();
}
```

```
//  Your own functions
move_arrow(){
                if (status_arrow == "down")
                {
                  status_arrow = "right";
                  arrow.setTargetX(211);
                  arrow.setTargetY(80);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
}

arrow.ontargetreached(){
                if (status_arrow == "down")
                {
                  status_arrow = "right";
                  arrow.setStartFrame(0);
                  arrow.setEndFrame(0);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(211);
                  arrow.setTargetY(80);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
                else if (status_arrow == "right")
                {
                  status_arrow = "up";
                  arrow.setStartFrame(0);
                  arrow.setEndFrame(1);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(241);
                  arrow.setTargetY(80);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
                else if (status_arrow == "up")
                {
                  status_arrow = "left";
                  arrow.setStartFrame(1);
                  arrow.setEndFrame(2);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(241);
                  arrow.setTargetY(40);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
                else if (status_arrow == "left")
                {
                  status_arrow = "down";
                  arrow.setStartFrame(2);
                  arrow.setEndFrame(3);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(211);
                  arrow.setTargetY(40);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
}
```

* Anim.m MAKI file for Complex Tutorial Skin

W3 MAKI Tutorial - Part IV. Skin Scripting (MAKI)

1. The Scripting Engine (MAKI)
2. MAKI Script Organization

1. Scripting Engine (MAKI)

Here are some quick notes on MAKI to get you started:

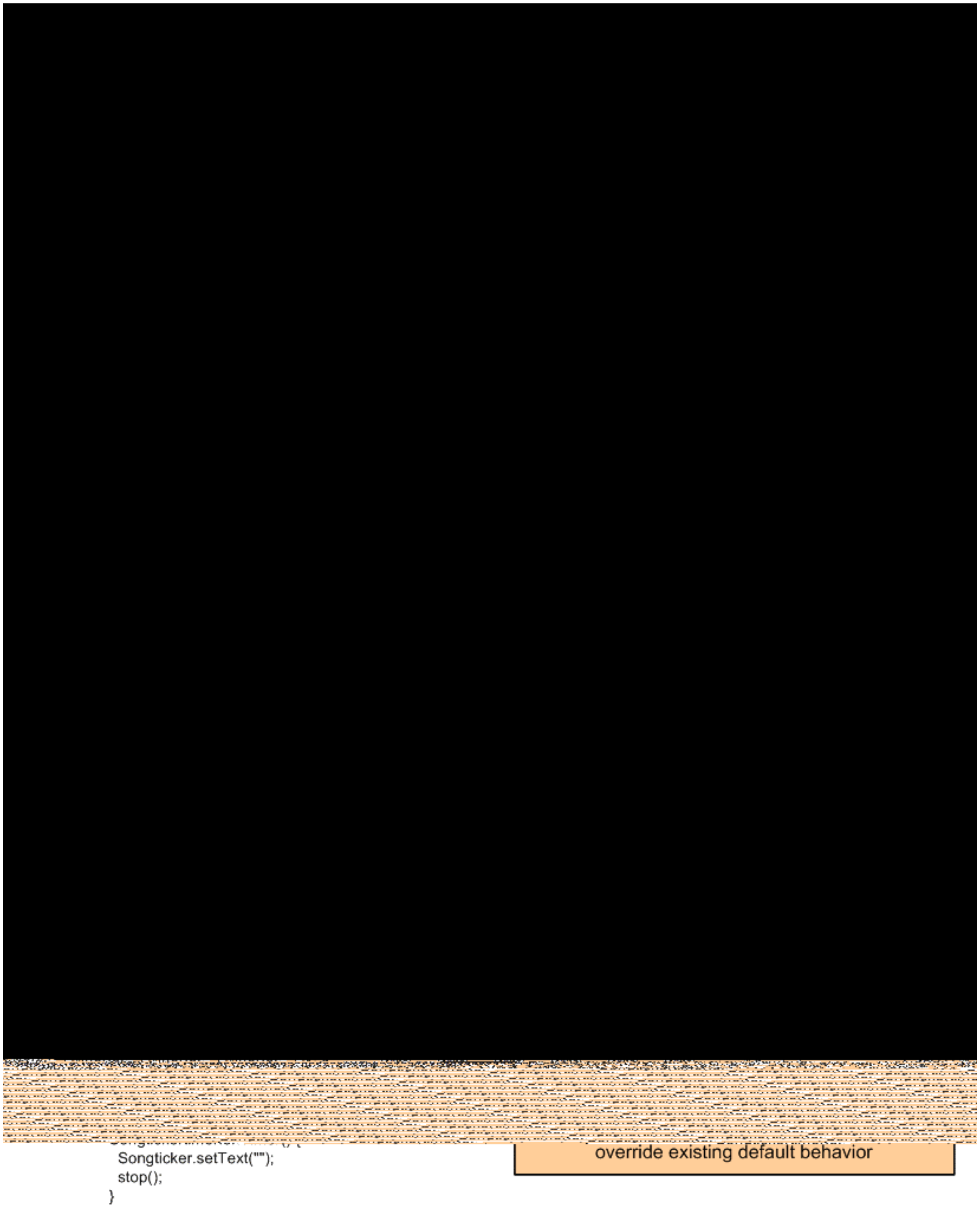
- MAKI stands for Make A Killer Interface.
- MAKI is a scripting engine for creating custom behavior from a Wasabi skin and set of components.
- MAKI scripts are precompiled into cross-platform bytecode. This means that the set of skin XML, elements and compiled scripts are completely cross-platform.
- Scripts can attach to any event of any object in the UI. Scripts can be loaded and unloaded in any order.

For more about MAKI, please read MAKI Overview.
(http://dev.winamp.com/wiki/Modern_Skin:_Maki_Overview)

2. MAKI Script Organization

Before we go over MAKI code, I'd like to go over the overall flow of a MAKI script. Please refer to the picture below. It shows what a typical MAKI script should look like.

- Part 1: Script Comments
- Part 2: #include Section
- Part 3: Variable Declaration
- Part 4: system.onUnloading() Section
- Part 5: system.onScriptLoaded() Section
- Part 6: Your own functions



Contact Us

Please email Ken Chen if you have any suggestion.

MAKI Tutorial - 4.1 Drawer Scripting Example

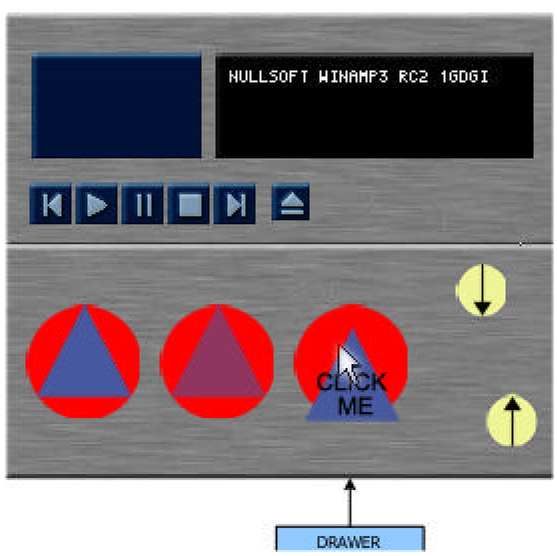
1. Scripting

What is a MAKI Script?
What is MAKI? **MAKI** stands for **M**ake **A** **K**iller **I**nterface. It is the scripting language for Winamp3. Scripting is used to produce the animations you see in the skin. Scripting is simple. It's easy! Anyone can do it! You'll love it.... Trust me.

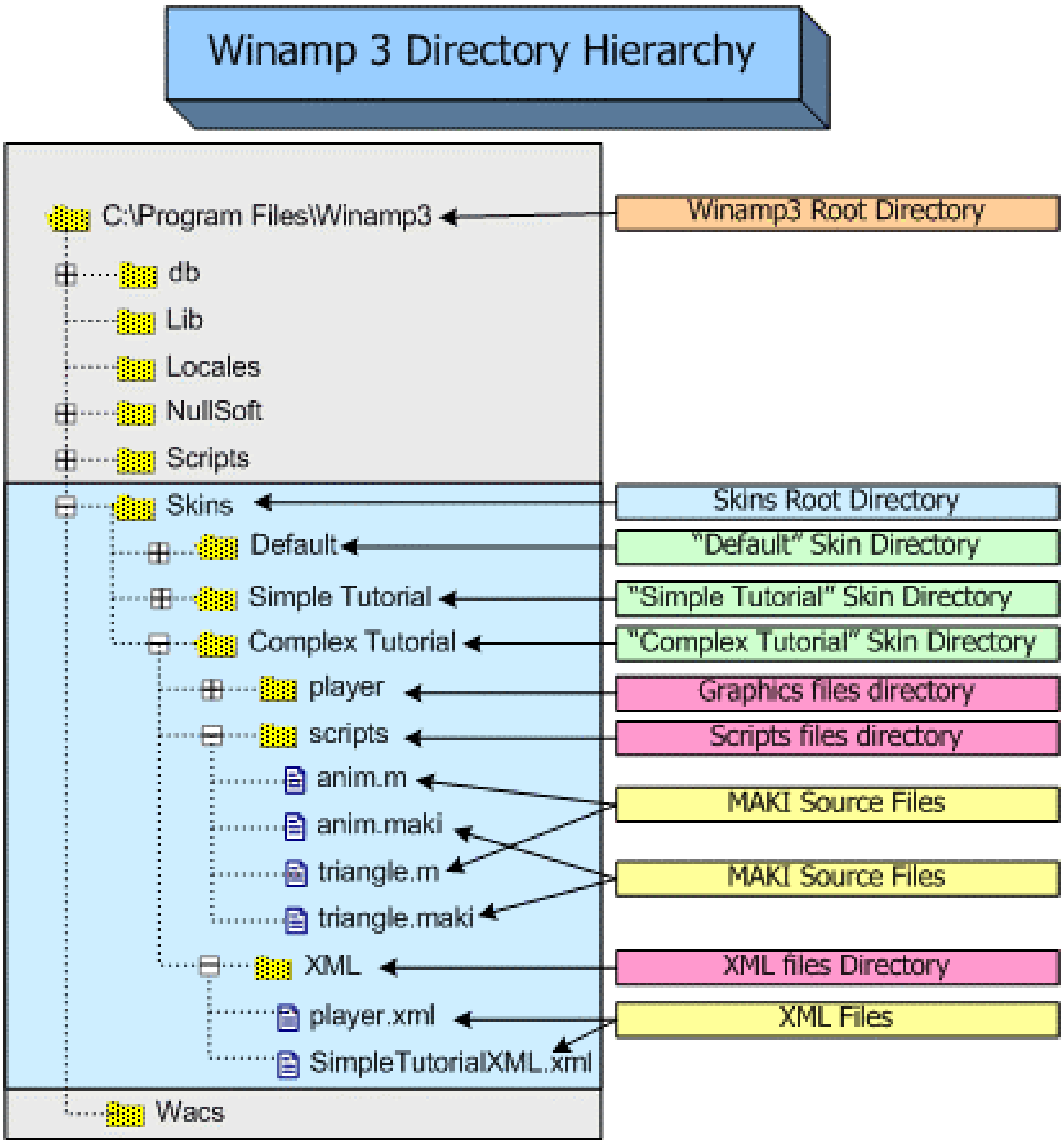
So how does a MAKI script work?
MAKI scripts are event based. In other word, your functions are used to define behaviors when an event occurs. If you don't define action for that event, the default behavior will be use d and the default behavior is defined by Winamp 3. Scripts are used to override an existing behavior. For example, when you click on the PL button, the playlist container appears. You can override that functionality so that the Media Library also appears. (However, we don't recommend you to do this)

What about elements I define in my own skins?
When you define new elements in your skin such as new buttons or sliders (elements that is not in the Default skin), essentially what you're doing is creating new o bjects. The behavior for that object is nothing.... nada until you specify an action in you XML or in your scripts.

The script to implement the animation you see below is short and simple. But it demonstrates the power of Winamp 3: with little programmin g, you can add functionalities to your skin or user interface. For more about MAKI, please read MAKI Overview.
(http://dev.winamp.com/wiki/Modern_Skin:_Maki_Overview)



2. Compiling a MAKI script



Here're the basic concepts that you should know before starting.

a) Directory hierarchy

All your scripts for your skin should be together in a subdirectory. Usually that subdirectory is called "scripts". In terms of hierarchy, that subdirectory is placed under your skin directory. In our example, "scripts" subdirectory is under the "ComplexTutorial" skin directory.

b) .m and .maki files

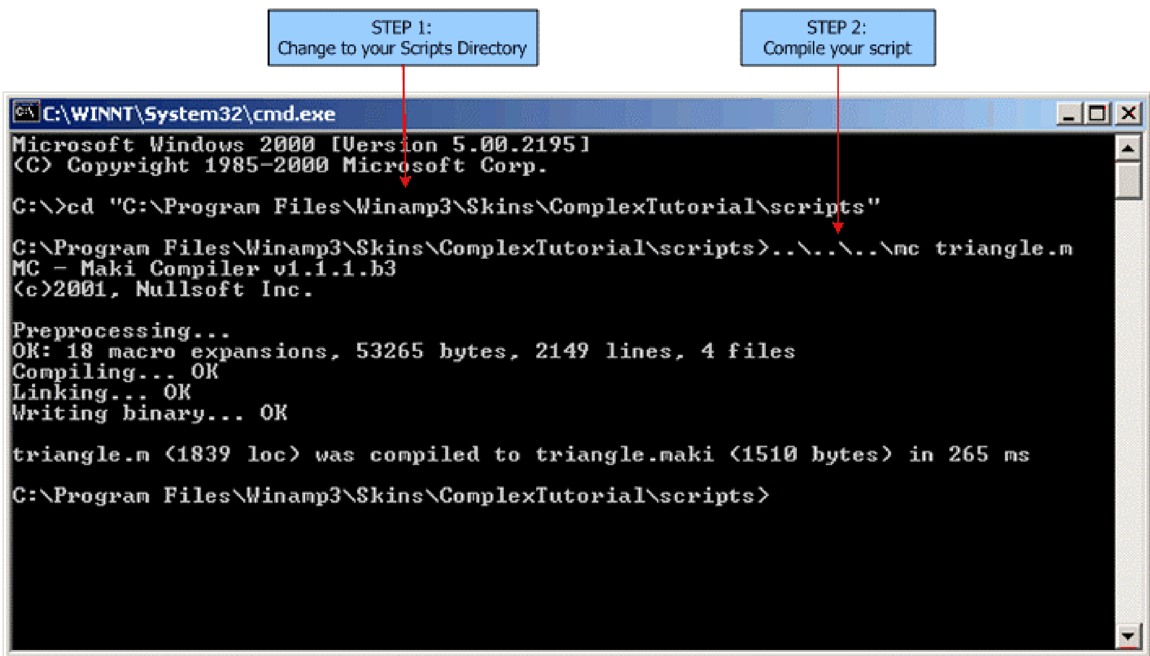
A sourcecode file is a text file that you can read and understand wit h a text program like Notepad or Editplus. The sourcecode file for your script has the .m extension. A compiled file is the result after being compiled by the compiler. It has been optimized (therefore unreadable for you) for the computer to run. The compiled script has the .maki extension.

c) Compiling a MAKI file

To compile a MAKI scripting, you must use the MAKI Compiler (mc.exe) in your Winamp3 directory. It is located in the same directory as your Winamp3 player. Chances are that it is located in "C:\Program Files\Winamp3\Studio.exe" directory.

1) Open a DOS Prompt: "Start Menu" => "Run..." => type "command"
2) Change path to script directory for your skin. For this example, the path is "C: \Program Files\Winamp3\Skins\ComplexTutorial\Scripts"
3) Run mc.exe on your script. Because mc.exe is not located in the same directory as your script, you need to type "..\..\..\mc" in order to run it. For our example, it is ".. \..\..\mc triangle.m" to compile triangle.m into triangle.maki.

You can set an environment variable to your Winamp3 directory so you don't have to type ".. \..\..\" to access mc.exe. You can just type "mc triangle.m" instead. (To set an environment variable: Right on "My Computer" => "Advanced" tab => Environment Variables.)



3. MAKI Script - triangle.m

The drawer animation is done through MAKI Scripting. The script is short and simple but it demonstrates the power of Winamp 3: with little programming, you can add functionalities to your skin or user interface. In your XML, you need t o specify which MAKI script you want to use.

```
<!-- Circle #3 Group Definition: Drawer -->
<groupdef id="complex.circle3" name="Red Circle 3">
  <layer id="redcircle3"
    x="144" y="30"
    ghost="0"
    image="complex.redcircle"
  />
  <script id="complex" file="scripts/triangle.maki"/>
  <button id="complex.circle3.triangle3"
    x="144" y="30"
    ghost="0"
    image="complex.triangle.clickme"
    move="0"
    rectrgn="0"
    sysregion="1"
  />                <!-- Y: 30 closed, 60 open -->
</groupdef>
```

In your Groupdef, you need to specify which MAKI script you want to use.

Lets break down the script:

a. Introduction Comments:

The first part of the script should be a brief introduction to the script. It should talk about the purpose and the functionality of the script.

```
//  Complex Tutorial Skin - triangle.m MAKI file
//  Created by Ken Chen for the official Winamp3 skinning site.
```

b. #include Section:

The second part is the #include section. Use "#include" to include any file you need. If you need to call or use another function that is defined in another file, make sure you include that file in your script.

```
//  #include Section
#include "../../../lib/std.mi"
#include "../../../lib/config.mi"
#include "../../../lib/pldir.mi"
```

c. Variable Declaration Section

Any variable that you'll use in your script, declare them in this section.

```
// Variable Declaration Section
global Button button_triangle;
global Int moving_triangle;
global String status_triangle;
```

d. System.onScriptUnloading() Section

When the script is unloaded, this part of the script will run. Typically, you do your clean up in this section. Our triangle script is simple and therefore doesn't need anything in this section.

```
// System.onScriptUnloading() Section
System.onScriptUnloading(){
}
```

e. System.onScriptLoaded() Section

When the script is first loaded, this part of the script will run. Typically, if anything you want to make happen when the script first start, you do that here. In our example, we initialized some objects and set the initial state for those objects. Just like the previous section, it is perfectly fine to leave this section empty. If you do that, nothing will happen when the script is loaded.

```
// System.onScriptLoaded() Section
System.onScriptLoaded(){
  group triangleDrawer = getScriptGroup();
  button_triangle = triangleDrawer.findobject("complex.circle3.triangle3");
  status_triangle = "closed";
  moving_triangle = 0;
}
```

f. You own functions

This section is where you put all your own functions that will define the behavior of your script. Make sure that any variable you use in this function is decared and their initial state has been initialized. For our example, we have two functions.

- button_triangle.onleftbuttonup(int x,int y) function

This function responds to the event when the triangle button is pressed. The triangle buttons has two states: open or closed. Depending on the current state, its position is changed so th at it is moved.

- button_triangle.ontargetreached() function

This function responds to the event when the triangle has reached its destination. What event occurs, the state of the button is changed to its new state.

```
// Your own functions
button_triangle.onleftbuttonup(int x,int y){
  if( status_triangle == "closed" && moving_triangle != 1 ){
                moving_triangle = 1;
                button_triangle.setTargetY(60);
                button_triangle.setTargetSpeed(0.25);
                button_triangle.gototarget();
  }else if( status_triangle == "open" && moving_triangle != 1){
                moving_triangle = 1;
                button_triangle.setTargetY(30);
                button_triangle.setTargetSpeed(0.15);
                button_triangle.gototarget();
  }
}

button_triangle.ontargetreached(){
  if( status_triangle == "closed" ){
                status_triangle = "open";
                moving_triangle = 0;
  }else if( status_triangle == "open" ){
                status_triangle = "closed";
                moving_triangle = 0;
  }
}
```

MAKI Tutorial - 4.2 Animating Through Script

1. Animating Skins
2. MAKI Script - anim.m

1. Animating Skins

An animated layer does not move. However, an animating skin does move. To move a graphical element in animating layer, you must write a MAKI script. In this section, we wi ll show you how to do a sequence of animation that you see in this skin.

2. MAKI Script - anim.m

Couple things you need to notice in your XML:

i) Both animated layer and animating skin are "AnimatedLayer" type.
ii) Both use the same graphical eleme nt.
iii) The difference lies that animating skin uses a MAKI script to move the arrow.

```

<!-- Circle #4 Group Definition: Animated Layer -->
<groupdef id="complex.circle4" name="Rotating Circle">
  <animatedlayer id="rotateCircle"
    x="225" y="10"
    w="25" h="25"
    image="complex.yellowcircle-arrow"
    autoplay="1"
    move="0"
    autoreplay="1"
    rectrgn="0"
    sysregion="1"
  />
</groupdef>

<!-- Circle #5 Group Definition: Scripting -->
<groupdef id="complex.circle5" name="Moving Circle">
  <script id="anim" file="scripts/anim.maki"/>
  <animatedlayer id="arrow"
    x="211" y="40"
    w="25" h="25"
    image="complex.yellowcircle-arrow"
    autoplay="0"
    move="0"
    autoreplay="0"
    rectrgn="0"
    sysregion="1"
  />
</groupdef>
```

- Both are Animatedlayers

- Both use the same graphic element

- For animating skin: MAKI script is required. Specify which script you're using.

Lets break down the script:

a. Introduction Comments:

The first part of the script should be a brief introduction to the script. It should talk about the purpose and the functionality of the script.

```
//  Complex Tutorial Skin - anim.m MAKI file
//  Created by Ken Chen for the official Winamp3 skinning site.

```

b. #include Section:

The second part is the #include section. Use "#include" to include any file you need. If you need to call or use another function that is defined in another file, make sure you include that file in your script.

```
//  #include Section
#include "../../../lib/std.mi"
#include "../../../lib/config.mi"
#include "../../../lib/pldir.mi"

```

c. Variable Declaration Section

Any variable that you'll use in your script, declare them in this section.

```
//  Variable Declaration Section
function move_arrow();

global String status_arrow;
global AnimatedLayer arrow;

```

d. System.onScriptUnloading() Section

When the script is unloaded, this part of the script will run. Typically, you do you r clean up in this section. Our triangle script is simple and therefore doesn't need anything in this section.

```
//  System.onScriptUnloading() Section
System.onScriptUnloading(){
}

```

e. System.onScriptLoaded() Section

When the script is first loaded, this part of the script will run. Typically, if anything you want to make happen when the script first start, you do that here. In our example, we initialized some objects and set the initial state for those objects. Just like the previous section, it is perfectly fine to leave this section empty. If you do that, nothing will happen when the script is loaded.

In this script, we set the initial state for several variables and call "move_arrow()" function when the script is loaded.

```
// System.onScriptLoaded() Section
System.onScriptLoaded(){
		group arrowAnim = getScriptGroup();
		arrow = arrowAnim.findobject("arrow");
		status_arrow = "down";
		move_arrow();
}
```

f. You own functions

To move the arrow, we use two functions. The first function, "move_arrow ()", is called by "System.onScriptLoaded()" when the script is loaded. The function of "move_arrow()" is to move the arrow down from its initial location. To do this, it sets the x and y coordinate and the speed for that movement. When the arrow reaches th e target location, an event is fired telling the system that the arrow has reached the target location.

Winamp3 is an event driven system. The second function is designed to handle the event when the arrow reaches its destination. Depending on its state, it will direct the arrow to move to a new location. This function is called over and over again every time the arrow reaches its destination. This simple function directs the movement of the arrow that you see in the skin.

```
//  Your own functions
move_arrow(){
                if (status_arrow == "down")
                {
                  status_arrow = "right";
                  arrow.setTargetX(211);
                  arrow.setTargetY(80);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
}

arrow.ontargetreached(){
                if (status_arrow == "down")
                {
                  status_arrow = "right";
                  arrow.setStartFrame(0);
                  arrow.setEndFrame(0);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(211);
                  arrow.setTargetY(80);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
                else if (status_arrow == "right")
                {
                  status_arrow = "up";
                  arrow.setStartFrame(0);
                  arrow.setEndFrame(1);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(241);
                  arrow.setTargetY(80);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
                else if (status_arrow == "up")
                {
                  status_arrow = "left";
                  arrow.setStartFrame(1);
                  arrow.setEndFrame(2);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(241);
                  arrow.setTargetY(40);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
                else if (status_arrow == "left")
                {
                  status_arrow = "down";
                  arrow.setStartFrame(2);
                  arrow.setEndFrame(3);
                  arrow.setSpeed(0);
                  arrow.setAutoReplay(0);
                  arrow.play();
                  arrow.setTargetX(211);
                  arrow.setTargetY(40);
                  arrow.setTargetSpeed(1.5);
                  arrow.gototarget();
                }
}
```