Guide to RCOmage and Animations

Learn the basics of XML in an RCO file.

A Reference for the Rest of Us!®

Table of Contents

Introduction
Requirements
1. Dumping with RCOmage
1.1 RCOmage dumping GUI
1.2 Dumping RCO files
2. Compiling with RCOmage
2.1 RCOmage Compiling GUI
2.2 Compiling RCO files
3. Structure of the XML
3.1 Trees and branches
3.2 Image Tree
3.3 Object Tree
4. Animations
4.1 Animation tree and animation types
4.2 Resize
4.3 Rotate
4.4 Fade Object
4.5 Re colour object
4.6 Move To
4.7 Fire Event
4.8 Delay
Author’s notes
Thanks
Contact

Introduction

In the past making a PSP theme would require editing the RCO file. Editing had its limitations to just replacing images for something that is desired. This was how it was done for many years and still is the main method. After some time RCOmage was created by no other then coding genius Zinga Burga who also brought about RCOeditor and several other useful tools to the PSP community. With the development and release of RCOmage, RCO files can now be broken down to its bare structure allowing users to manipulate RCO files even further then just replacing images. But before we jump into manipulating the RCO file we must understand some basics of RCOmage and what the RCO file consists of.

In general once the RCO file has been decompiled, you will have the contents of the RCO file and the XML (Extensible Markup Language). The XML file will be the bread and butter of editing RCO files. It will allow you to add or delete images, as well as animations and change functions of the RCO file. We must keep in mind that editing can only go so far because there are still unknown values in the XML which limit our manipulation and we only have so much knowledge of what tags or commands can be used in the XML. Without holding you back anymore I will begin with the basics of using RCOmage all the way through to animating.

NOTE: Please note that I will not be going through everything in the XML, I will only go through the most important areas of the XML as this will save you time from reading unnecessary things

Requirements

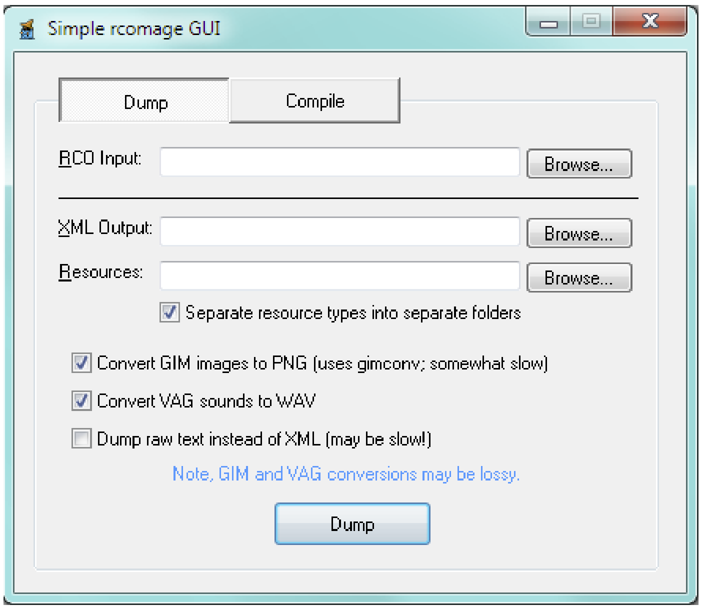
Just before we dive into the whole thing, we are going to need several tools to make things go smoothly and work. The tools that I will be using and recommending are RCOmage v1.0.2, Notepad++ and decrypted RCO files. I'll provide you with the links so you won't have to go looking around for them.

RCOmage - RCOmage v1.0.2
RCO files - Decrypted RCO files
Notepad++ (This is free so it is the direct link to the site) - Notepad++

1. Dumping with RCOmage

1.1 RCOmage dumping GUI

Let's begin our tutorial starting with using RCOmage and understanding its functions. Start off by executing RCOmage. You will be presented with a simple GUI with several options (**Fig .01**).



[✓] Separate resource types into separate folders

[✓] Convert GIM images to PNG (uses gimconv; somewhat slow)

[✓] Convert VAG sounds to WAV

[ ] Dump raw text instead of XML (may be slow!)

Note, GIM and VAG conversions may be lossy.

(Fig .01)

The option dump will automatically be selected first for you.

RCO Input - is where you will browse to your desired RCO file to be dumped.

XML Output - is the directory of where you want the XML sheet to be outputted.

Resources - is the directory of where you want the images, sound etc to be outputted (this is chosen automatically for you but you can change it if you want, I would not recommend that though).

Separate resource types into separate folders – this will sort out the different file types into different folders. E.g. Images, sound, text etc.

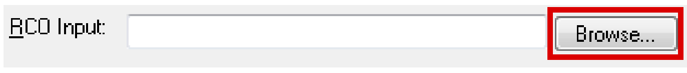
Convert GIM images to PNG – converts the original .GIM file format into .PNG.

Convert VAG sounds to WAV – converts the original sound file VAG to WAV.

Dump raw text instead of XML – not sure about this. I still get a XML file.

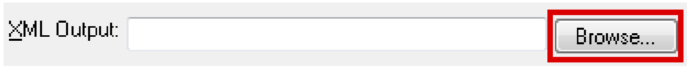
1.2 Dumping RCO files

To dump a RCO file we begin by selecting the "Browse..." button that is on the same line as RCO Input (**fig .02)** and select a file that you wish to edit.



(Fig .02)

Next we select where we want the XML file to be outputted. Same as the step above, select "Browse..." **(Fig .03)** choose a destination and a file name for the XML file (I recommend just name it the same as the RCO file so you won't get confused).



(Fig .03)

The next step is optional; this is choosing the directory where the images, sounds etc will be extracted. The directory is automatically generated for you when you save the XML output so I won't go into it.

The checkboxes are all optional depending on how you like to work.

If the "Separate resource types into separate folders" is checked then each file type will have its own folder after extraction. If not, then all files will be in one folder.

The next two checkboxes are obvious, it does what it says. I normally have these unchecked because it saves time not converting. Totally up to the user.

Dump raw text? I just get XML files like normal dumping so nothing to say here.

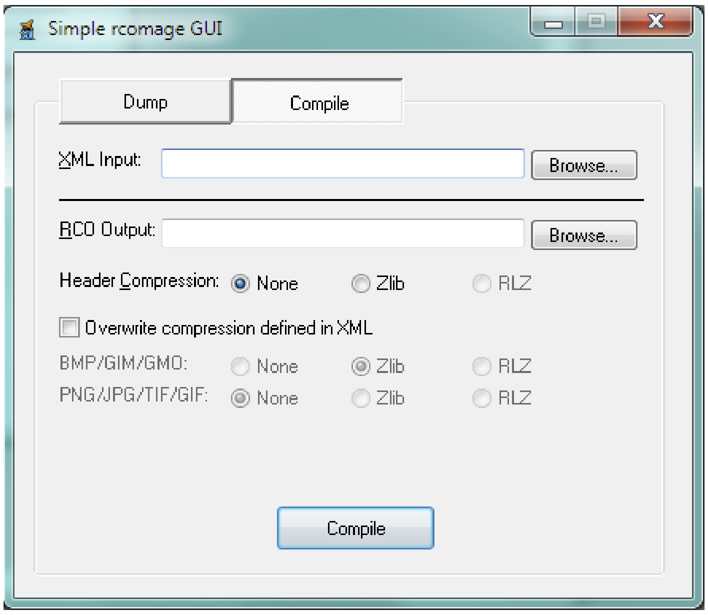
Once you are satisfied with your options it is just a simple click of "Dump". Once the process is done you will get a message of "RCO successfully dumped".

Congratulations, you have now successfully dumped your first RCO file.

2. Compiling with RCOmage

2.1 RCOmage Compiling GUI

This next section will be on the compiling functions.
Firstly execute RCOmage; once again you will be presented with the GUI with the dump option selected. Instead of dumping select "Compile" (**fig .04**).



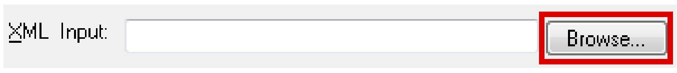
XML Input:
RCO Output:
Header Compression: [✓ None] Zlib RLZ
☐ Overwrite compression defined in XML
BMP/GIM/GMO: None [✓ Zlib] RLZ
PNG/JPG/TIF/GIF: [✓ None] Zlib RLZ

(Fig .04)

Similar to the "Dump" screen, you will have several options.
XML Input – is the XML file that you have saved and ready to be complied.
RCO Output – is the directory of where your newly created RCO file will be saved.
Header Compression – This compresses the headers in some RCO files otherwise it cannot be used. An example would be the file "osk_plugin_500.RCO", when compiling make sure to select "Zlib" otherwise it will not load properly on the PSP.
Overwrite compression defined in XML – This allows us to choose what kind of compression will be used for image file types. The XML file will have a predefined compression but this allows us to force a different kind.

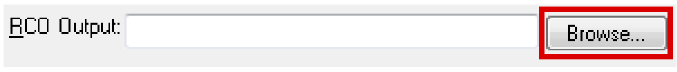
2.2 Compiling RCO files

Compiling RCO files is straight forward and similar to decompiling a RCO file.
Start by selecting "Browse..." on the same line as XML Input **(fig .05)** and locate your saved XML file.



(Fig .05)

Next we will browse for a directory to save our newly created RCO file. We do this by selecting "Browse..." on the same line as RCO Output **(fig .06).**



(Fig .06)

Header compression, I find that I only use that for compiling the file "osk_plugin_500.RCO". So do select "Zlib" if you are compressing that particular file or any other that needs it.

Check the overwrite compression defined in XML if needed; if not then just leave the check box blank.

Once you are done with selecting your options, click "Compile" and you will have successfully created your newly edited RCO file.

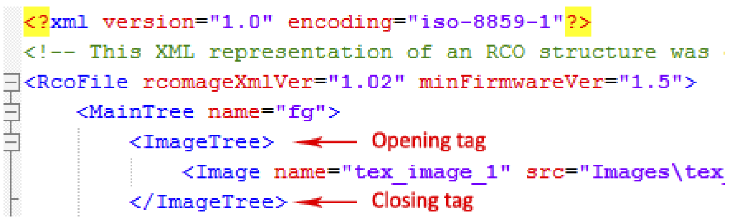
NOTE: You can get error messages when compiling. This can be due to several things such as mistakes in the coding of your XML file and/or bad text files when dumping. These error messages can be overcome; lucky for us Zinga Burga has coded RCOmage to indicate which line the error it is occurring. It is just a matter of looking back at the coding and finding those mistakes.

3. Structure of the XML

3.1 Trees and branches

We will now go onto understanding the structure of the XML and what it consists of. Within the XML file there will be several “trees”; these “trees” will be the base of our XML. Within these “trees” we will have (what I call) “branches”, these “branches” will be vital to our tree because these “branches” will hold our information to build our RCO file. Just imagine a tree without branches and only a tree trunk, not so much of a tree now is it?

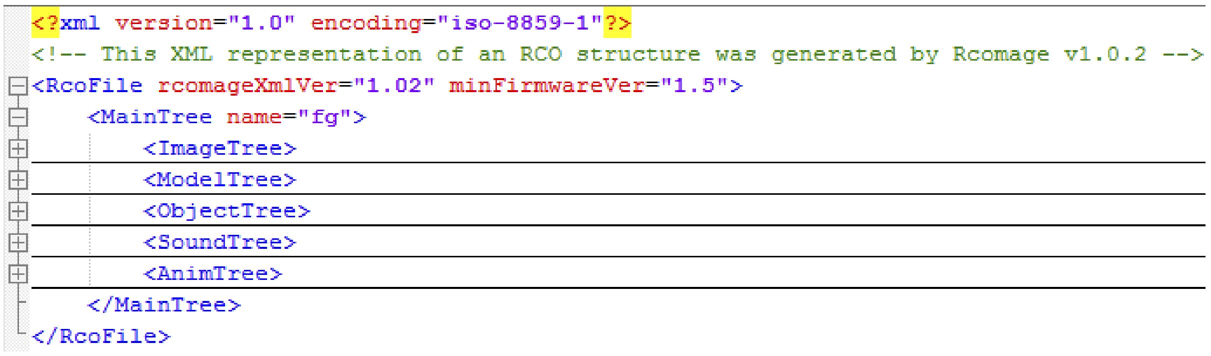
As I mentioned earlier, there are several trees in the XML file. These trees are: Main Tree, Image Tree, Model Tree, Sound Tree, Object Tree and Animation Tree. Not all trees are used in an RCO file. Some trees can actually be omitted and not used at all depending on the RCO file. Most important thing to remember is that when creating trees you must always close your tags. If not then you will be getting error messages when compiling. Here is what it should look like in **figure .07**.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- This XML representation of an RCO structure was
<RcoFile rcomageXmlVer="1.02" minFirmwareVer="1.5">
    <MainTree name="fg">
        <ImageTree>
            <Image name="tex_image_1" src="Images\tex
        </ImageTree>
```

(Fig .07)

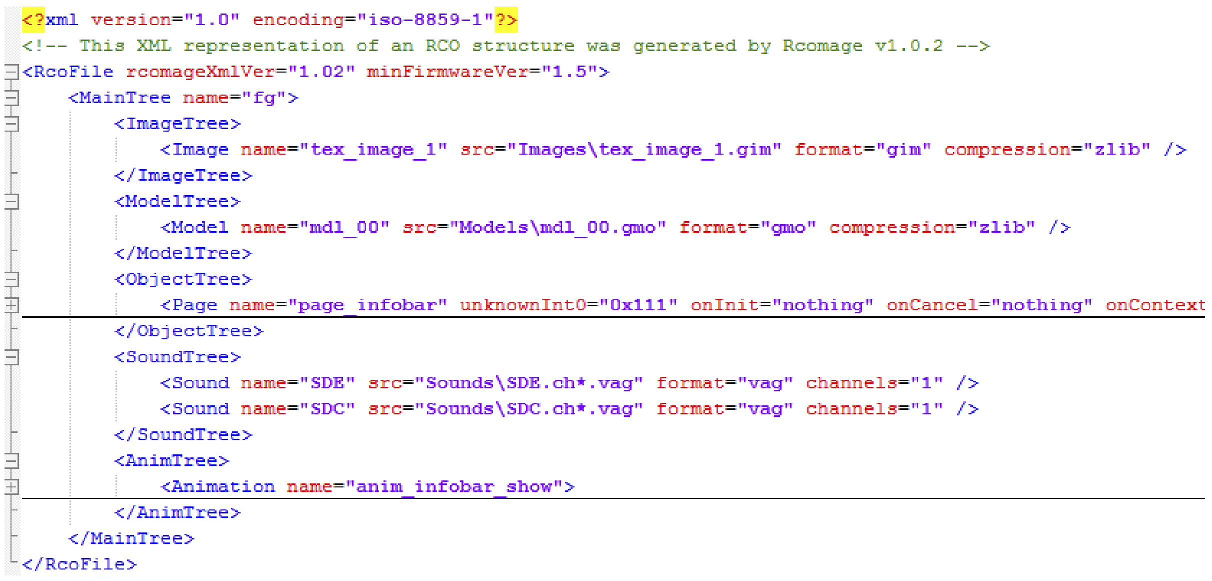
Below you can get a general idea of what the XML could look like with all the trees **(fig .08)**.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- This XML representation of an RCO structure was generated by Rcomage v1.0.2 -->
<RcoFile rcomageXmlVer="1.02" minFirmwareVer="1.5">
    <MainTree name="fg">
        <ImageTree>
        <ModelTree>
        <ObjectTree>
        <SoundTree>
        <AnimTree>
    </MainTree>
</RcoFile>
```

(Fig .08)

Here is a general idea of what the XML could look like with all the branches in the trees **(fig .09)**.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- This XML representation of an RCO structure was generated by Rcomage v1.0.2 -->
<RcoFile rcomageXmlVer="1.02" minFirmwareVer="1.5">
    <MainTree name="fg">
        <ImageTree>
            <Image name="tex_image_1" src="Images\tex_image_1.gim" format="gim" compression="zlib" />
        </ImageTree>
        <ModelTree>
            <Model name="mdl_00" src="Models\mdl_00.gmo" format="gmo" compression="zlib" />
        </ModelTree>
        <ObjectTree>
            <Page name="page_infobar" unknownInt0="0x111" onInit="nothing" onCancel="nothing" onContext
        </ObjectTree>
        <SoundTree>
            <Sound name="SDE" src="Sounds\SDE.ch*.vag" format="vag" channels="1" />
            <Sound name="SDC" src="Sounds\SDC.ch*.vag" format="vag" channels="1" />
        </SoundTree>
        <AnimTree>
            <Animation name="anim_infobar_show">
        </AnimTree>
    </MainTree>
</RcoFile>
```

(Fig .09)

Now all of that does look quite confusing and scary at first but it will all become clear once I go into the important details needed to understand and create your own RCO files.

3.2 Image Tree

The Image Tree, of course the name says it all, it houses all of our branches that will hold our images for the RCO file. Our Image Tree will use the tag of <ImageTree> to open and </ImageTree> to close. Now we will take a look at the branch within the tree **(fig .10).**

```
<ImageTree>
    <Image name="tex_image_1" src="Images\tex_image_1.gim" format="gim" compression="zlib" />
</ImageTree>
```

(Fig .10)

Name - is of course the name you give your image, it is important to give it a proper name and not something too generic because it can create errors when compiling, also important because the name will be used to link up to other trees and branches. This name does not have to be the same as the actual file itself but can be different.

Src - is the source of your images, the XML will have a default path of where you extracted the RCO file so you should not need to change it, the only thing you need to do is when using your own images remember to place them into the “Images” folder. Remember what name and file format of your image.

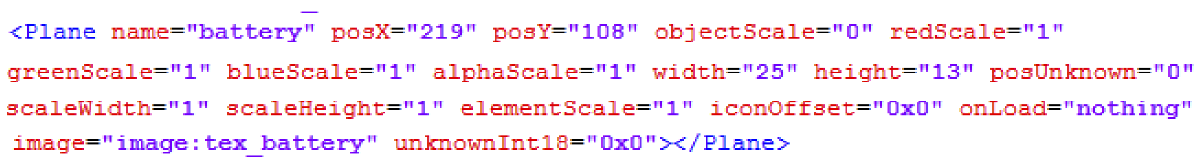
Format - is what the image file will be converted into once the RCO file is compiled. Images in RCO files are in GIM format so there is no need to change this.

Compression - is what type of compression will be used on the file. RCO files have a zlib compression and are best to be left as is.

I will now follow up with a simple example of how to use the Image Tree with **figure 10**.
I will open up the branch with an “<Image” tag. As you can see my image has the name of “tex_image_1”. My source image would be named “tex_image_1.gim” but if you have a different format like PNG you would name it “tex_image_1.png”. Remember to put your image file into the “Images” folder or else when compiling you will get an error. Format and compression should not need to be changed. Last but not least you will need to remember to close the branch with “/>”

3.3 Object Tree

Our object tree has the tag of <ObjectTree>. The object tree will help us store information in our branches about our images such as; size, opacity, position and functions. It will also help us recognise text messages we read in the XMB. Most importantly this tree will display our images and text. Without this tree then we will not be able to see images or text. There is a lot of information which can be stored in each of our object branches because there are several types of branches. These branches are: Page, Plane and Text. I will be going through the features of the Plane branch since it is the most used. First things first, the Plane branch can be a part of another branch within the Page branch or independent. It does not really matter how you put it, either way it can still work as long as you give it the proper information. Underneath is an example of what it could look like **(fig .11)**, I've actually put it into several lines since it is quite long.



```
<Plane name="battery" posX="219" posY="108" objectScale="0" redScale="1"
greenScale="1" blueScale="1" alphaScale="1" width="25" height="13" posUnknown="0"
scaleWidth="1" scaleHeight="1" elementScale="1" iconOffset="0x0" onLoad="nothing"
image="image:tex_battery" unknownInt18="0x0"></Plane>
```

(Fig .11)

Name – is the name of the object.

posX and posY – is the position of the object on screen. posX refers to the position on the X axis and posY refers to the position on the Y axis.

redScale, greenScale and blueScale – is the colour for your object. It is normally all default at 1. It is best not to play with this and just colour your objects manually. If you do want to experiment with these values then you have to obtain the colours value between 1 -255 and divide by 255 to obtain a decimal number.

alphaScale – is the transparency of the object. A value of 1 being fully visible and a value of 0 not being visible.

width and height – is the dimensions of your object.

onLoad – is what this object is to do when the RCO file is loaded.

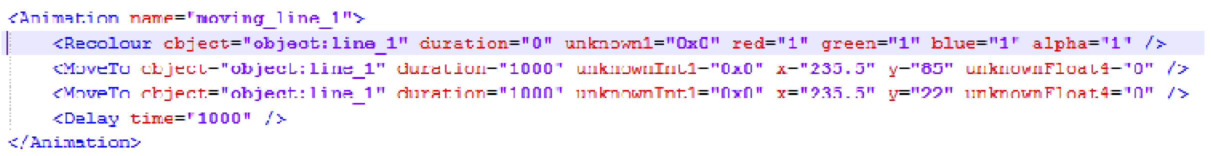
image – is the image it is to be loaded when the RCO file is loaded. If it is linked to an image it must be linked back to the image branch with the proper image name.

Now an example with **figure 11**. Opening the branch with "<Plane" and remembering to close it with "></Plane>". My object name is "battery" (which is the battery icon), with the X and Y positions of 219 and 108 respectively. AlphaScale is set to 1, otherwise we would not be able to see it. OnLoad is set to "nothing", this just means that this particular branch won't load anything else in the RCO file. In other cases onLoad can be used to load animations, we will get to this later on. My image is set to "image:tex_battery", the word "image" before the ":text_battery" indicates that the image is to come from our image tree. The image in our case is the image "tex_battery"; this must mean that in the image tree there is an image branch that is called "tex_battery".

4. Animations

4.1 Animation tree and animation types

A tag of <AnimTree> is used for our animation tree. Our animation branches will have an opening tag of "<Animation", followed by a "name" and ending the branch with a ">", this is not closing the branch. When we finish the animation, we will close the branch with "</Animation>". You can get a general idea from below **(fig .12).**



```
<Animation name="moving_line_1">
    <Recolour object="object:line_1" duration="0" unknown1="0x0" red="1" green="1" blue="1" alpha="1" />
    <MoveTo object="object:line_1" duration="1000" unknownInt1="0x0" x="235.5" y="85" unknownFloat4="0" />
    <MoveTo object="object:line_1" duration="1000" unknownInt1="0x0" x="235.5" y="22" unknownFloat4="0" />
    <Delay time="1000" />
</Animation>
```

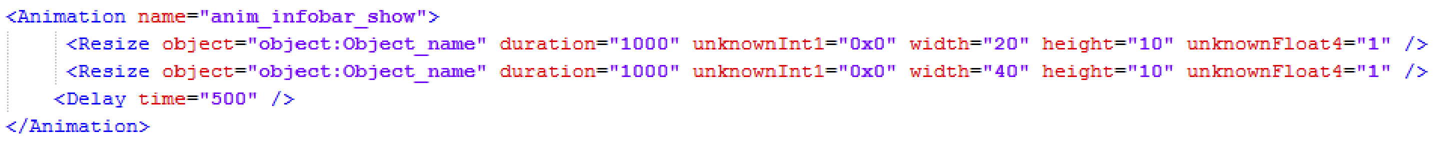
(fig .12)

There are many tags that are involved in animating objects in an RCO file. These are: Resize object, Rotate, Fade object, Re colour, Move object, Fire Event and Delay.

I will now go through every single tag listed above followed by an example.

4.2 Resize

Resize obviously means to change the size of the original image to something smaller or larger, depending on the values you input. Underneath is an example **(fig .13)**.



```
<Animation name="anim_infobar_show">
     <Resize object="object:Object_name" duration="1000" unknownInt1="0x0" width="20" height="10" unknownFloat4="1" />
     <Resize object="object:Object_name" duration="1000" unknownInt1="0x0" width="40" height="10" unknownFloat4="1" />
    <Delay time="500" />
</Animation>
```

(fig .13)

object – this will help us direct our animation to our object that is to be animated. The "object" before ":object_name" is to tell the animation that the object is to come from the object tree with that particular name.

duration – is the amount of time the object will have to undergo the animation. This will also determine the speed of the animation. A large value will result in a slow animation where's a small value will result in a fast animation.

delay - is a wait time where no action is to be done.

width and height – is the dimensions of the object. This is the resized dimensions of the object and not necessarily the dimensions of the actual object itself.

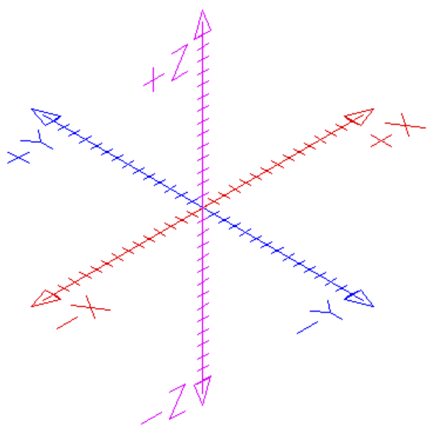
An example from **figure .13**, we can see that our object width and height is 20 and 10 respectively. Our resize tag dimensions do not have to be the same as the object's dimensions but it is best to keep the first resize tag the same as our objects. This means that our object's dimension is also the same in width and height. Our duration is 1000, this means that it takes a time frame of 1000 to reach the next resize tag. So during this time frame of 1000 it will resize itself to a width of 40 and height of 10. After the resize, we must have a delay time, in our case it is a time frame of 500. This is just a wait time where no action is to be done. We can now close the tag.

4.3 Rotate

Now rotate sounds pretty simple but it is more complicated then what is really is. There is the X, Y and Z axis to play with when rotating objects. This means that we could be working with 3 dimensional rotations depending if you want to apply it. Below is as an example of the branch **(fig .14)** as well as what the X, Y and Z axis **(fig .15)** looks like. As you can see it looks quite tricky when thinking about rotating objects. X and Y will rotate things on an angle. Z will rotate objects in a clockwise or anticlockwise fashion. This is because the Z axis is what we are facing, unlike the X and Y axis, which is on an angle. I won't be going into detail about X and Y rotations since it is quite complicated and I have not yet fully tried out this animation.

```
<Animation name="rotate">
    <Rotate object="object:wave_1" duration="1000" unknown1="0x0" x="0" y="0x0" z="-5" />
    <Delay time="0" />
    <FireEvent event="anim:rotate" />
</Animation>
```

(fig .14)



(fig .15) Imagine that this is the front of the PSP screen.

object – this will help us direct our animation to our object that is to be animated. The "object" before ":wave_1" is to tell the animation that the object is to come from the object tree with that particular name.

duration – is the amount of time the object will have to undergo the animation. This will also determine the speed of the animation. A large value will result in a slow animation where's a small value will result in a fast animation.

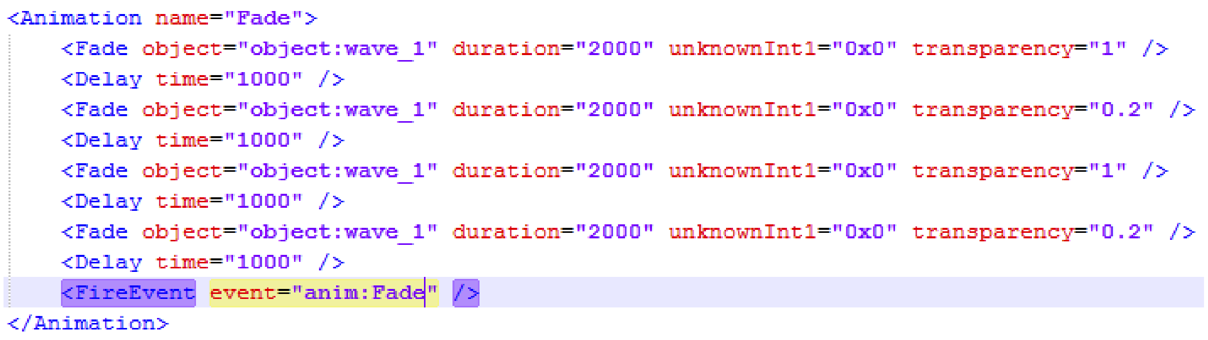
delay - is a wait time where no action is to be done.

X, Y and Z – are the axis which can be used to rotate objects.

Followed by an example with **figure .14**. With duration of 1000, our object must undergo a rotation of -5 on the Z axis. So my object will be rotating clockwise. A negative value will make the object rotate clockwise where's a positive value will make the object rotate anti clockwise. Here is the tricky part, the duration and axis value work together to determine rotation speeds. It is usually small values which will give fast rotations and large values which give slower rotations. Be sure to experiment and get the desired result.

4.4 Fade Object

Fade object allows us to make an object fade to a certain degree of transparency, ranging from a value of 1 to 0 and any decimal value in between. **Figure .16** shows us what the branch could look like.



```
<Animation name="Fade">
    <Fade object="object:wave_1" duration="2000" unknownInt1="0x0" transparency="1" />
    <Delay time="1000" />
    <Fade object="object:wave_1" duration="2000" unknownInt1="0x0" transparency="0.2" />
    <Delay time="1000" />
    <Fade object="object:wave_1" duration="2000" unknownInt1="0x0" transparency="1" />
    <Delay time="1000" />
    <Fade object="object:wave_1" duration="2000" unknownInt1="0x0" transparency="0.2" />
    <Delay time="1000" />
    <FireEvent event="anim:Fade" />
</Animation>
```

(fig .16)

duration – is the amount of time the object will have to undergo the animation. This will also determine the speed of the animation. A large value will result in a slow animation where's a small value will result in a fast animation.

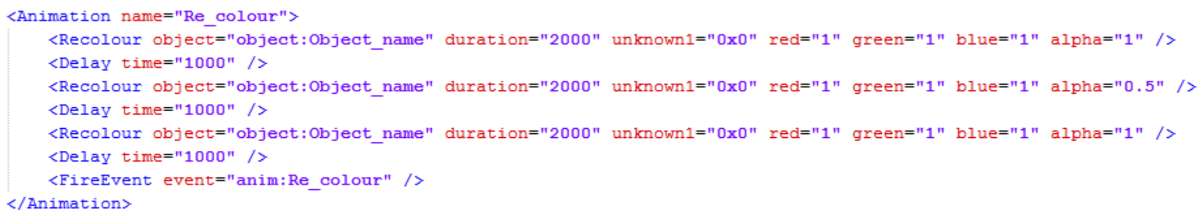
delay - is a wait time where no action is to be done.

transparency – is the degree of visibility of the object. A value closer to 1 will result in a more visible object where's a value closer to 0 will result in a less visible object.

We can see in **figure .16** that our first tag will start off with a value of 1, our object is completely visible. After a delay time of 1000 our object will only be visible with a transparency value of 0.2. It will then go on to repeat the process.

4.5 Re colour object

Re colour object is quite similar to fade object because it also deals with an objects transparency. The only noticeable difference is that instead of fading to the transparency value it will instantly make the object appear with the transparency value. Here is an example in **figure .17**.



```
<Animation name="Re_colour">
    <Recolour object="object:Object_name" duration="2000" unknown1="0x0" red="1" green="1" blue="1" alpha="1" />
    <Delay time="1000" />
    <Recolour object="object:Object_name" duration="2000" unknown1="0x0" red="1" green="1" blue="1" alpha="0.5" />
    <Delay time="1000" />
    <Recolour object="object:Object_name" duration="2000" unknown1="0x0" red="1" green="1" blue="1" alpha="1" />
    <Delay time="1000" />
    <FireEvent event="anim:Re_colour" />
</Animation>
```

(fig .17)

duration – is the amount of time the object will have to undergo the animation. This will also determine the speed of the animation. A large value will result in a slow animation where's a small value will result in a fast animation.

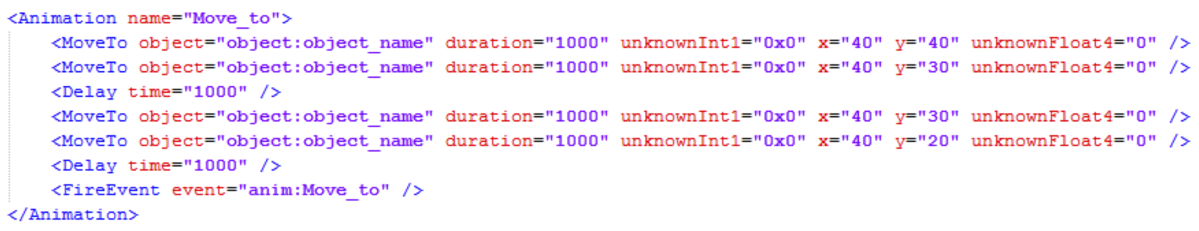
delay - is a wait time where no action is to be done.

alpha – is the degree of visibility of the object. A value closer to 1 will result in a more visible object where's a value closer to 0 will result in a less visible object.

Taking **figure .17** as an example, with an alpha value of 1, our object is completely visible. After the duration and delay time, our object will appear as only half visible with a value of 0.5. It will then go on to be completely visible again.

4.6 Move To

Move to is as obvious as it can get, simply moving our object on the screen by manipulating the values given to X and Y. This gives us the ability to not only move objects up and down but even diagonally, that is if you apply to the correct values. Below is an example in **figure .18.**



```
<Animation name="Move_to">
    <MoveTo object="object:object_name" duration="1000" unknownInt1="0x0" x="40" y="40" unknownFloat4="0" />
    <MoveTo object="object:object_name" duration="1000" unknownInt1="0x0" x="40" y="30" unknownFloat4="0" />
    <Delay time="1000" />
    <MoveTo object="object:object_name" duration="1000" unknownInt1="0x0" x="40" y="30" unknownFloat4="0" />
    <MoveTo object="object:object_name" duration="1000" unknownInt1="0x0" x="40" y="20" unknownFloat4="0" />
    <Delay time="1000" />
    <FireEvent event="anim:Move_to" />
</Animation>
```

(fig .18)

duration – is the amount of time the object will have to undergo the animation. This will also determine the speed of the animation. A large value will result in a slow animation where's a small value will result in a fast animation.

delay - is a wait time where no action is to be done.

X and Y – are the axis which can be used to move objects.

The first tag has an X and Y value of 40 and 40 respectively. The next tag will have our object staying in the same place in the X direction but going from 40 to 30 in the Y direction. An important thing to remember is that when you go onto the next tag after a delay you must start with the value of where the previous point was. So like in our example **(fig .18)**, our third tag is to have the same value X and Y of 40 and 30 as our second tag, the one just above.

4.7 Fire Event

This is quite a special tag, Fire event allow us to start up other animations in the animation tree or even just make an animation repeat itself upon finishing. I will take the example used from re colouring section **(fig .19)**.

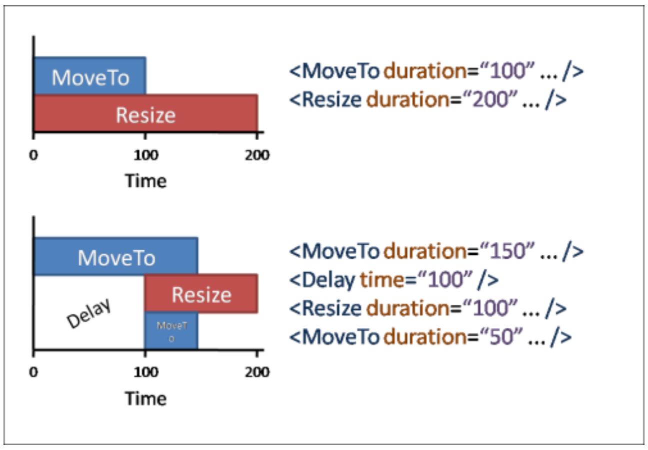
```
<Animation name="Re_colour">
    <Recolour object="object:Object_name" duration="2000" unknown1="0x0" red="1" green="1" blue="1" alpha="1" />
    <Delay time="1000" />
    <Recolour object="object:Object_name" duration="2000" unknown1="0x0" red="1" green="1" blue="1" alpha="0.5" />
    <Delay time="1000" />
    <Recolour object="object:Object_name" duration="2000" unknown1="0x0" red="1" green="1" blue="1" alpha="1" />
    <Delay time="1000" />
    <FireEvent event="anim:Re_colour" />
</Animation>
```

(fig .19)

After all of the tags have been animated comes our fire event. In our case it is "anim:Re_colour", this indicates that the RCO file will load up the animation in the animation tree called "Re_colour". Our object will then go through the same animation process over again without stopping.

4.8 Delay

Delay is not quite yet understood why it is needed, but it is indefinitely needed no matter what. Delay times are needed after tags, sometimes it is possible to have two of the same tags then comes a delay. Basically, the PSP executes all the commands concurrently until it hits a delay command. Here's a diagram which hopefully explains how it works **(fig .20)**:



```
<MoveTo duration="100" ... />
<Resize duration="200" ... />
```

```
<MoveTo duration="150" ... />
<Delay time="100" />
<Resize duration="100" ... />
<MoveTo duration="50" ... />
```

(fig .20)

Sometimes it must be only one type of tag then comes a delay. **Figure .18** demonstrates two of the same tags of "MoveTo" and after that comes a delay. **Figure .17** shows a single tag of "Recolour" and after that comes a delay. Try and become familiar with what is possible, what is not and experiment a little.

Authors notes

This brings the guide to a close. Hopefully this guide will help those who have not yet had time to fully study and understand the XML within an RCO file. There is still a lot that I have not yet fully discovered yet about the animation tags and hopefully will get more time to experiment then later on add more to this guide. But I feel that this guide is more than enough to get people started and later on release great themes.

Thanks

www.consolespot.com – where I learnt most of my knowledge
www.endlessparadigm.com – great supply of editing tools
www.psp-hacks.com – up to date information
Notepad++ - great free XML editing software
Team GEN – for continuing the CFW where DAX left off
Sony – for making the PSP
Dark Alex – for starting the CFW
Zinga Burga aka Zing Burg – brilliant software to make PSP themes easier
Vegetano1 – providing great resources to learn from
Highboy - brilliant software to make PSP themes easier
PatPat – CTFtool GUI
Anyone and everyone who helped me out with my current up to date knowledge about themes.
And anyone else that I forgot to mention.

Contact

If you would like to contact me with any queries, mistakes or additional info I can add to this guide please feel free to email me. Just keep in mind to use proper English with sentencing and grammar and also have a proper subject title in the email. I know that English is not a lot of people’s first language but please do try and use proper capitals and full stops, it makes it easier to understand.

xeroxidous@gmail.com