A Simplified Explanation of the Repository Technique

The Repository Technique Explained

I can hear you saying “the repository what!?"

The repository technique, also known as master/slave or linked files, and commonly used to make “Maxis match” custom meshes, is my favourite little trick to use when making items for the Sims 2.

A “repository” is a storage place; in this case, a base (master) mesh that contains the textures that will be used by multiple linked (slave) files. In the Sims, there are a wide variety of objects that use this technique. For example, all Maxis beds pull their bedding textures from a single base bed; the “Club Distress” armchair and loveseat use the texture from the “Club Distress” sofa, and so forth.

Its single biggest benefit is that you only need to recolour the repository object to recolour all of the objects that use its textures. It also cuts down (in some cases pretty significantly!) on how many textures your graphics card needs to hold in memory.

Example:
You have a set of 5 meshes with a total of 10 recolours available for each. They all use the same texture, but since you don’t know the repository technique you’ve set up each item to contain the “default” texture and then created 9 recolour packages for each mesh. You think you “only have 10 textures”, but what you’ve actually just given the game to handle is in fact fifty textures, eating up memory and processing time. And not only is it 50 textures for your graphics card to manage, it’s also 50 files for the game to load, index, and keep track of, because you have:

\[5 \text{ mesh files} + (5 \text{ meshes} \times 9 \text{ recolours per mesh}) = 5+45 = 50\]

Then you learn the repository technique and change it so one of your meshes is the master mesh. It contains 1 default texture and you make 9 recolours for it. The remaining 4 meshes pull their textures from the repository mesh.

Suddenly you only have 10 textures and a total of 14 files!

\[5 \text{ mesh file} + (1 \text{ master mesh} \times 9 \text{ recolours for that mesh}) = 5+9 = 14\]

Isn’t math fun? The difference between 50 and 14, between 10 textures and 50 textures, adds up significantly once you multiply it by the hundreds or even thousands of downloads you might have, which you are asking your PC to process and your graphics card to deal with the textures for. Personally, I’d rather have my graphics card redrawing my screen faster then having to juggle umpteen identical copies of a single texture.

So now you know why “matching” objects that use duplicate-texture recolours instead of the repository technique are a pet peeve of mine. Now… on to the meat and bones of how to actually set up objects to use this lovely (and surprisingly simple!) technique.
Using the Repository Technique

Necessary Software and the Example Mesh

You will need a copy of SimPE. This tutorial is aimed at users who already understand how to create their own custom objects, and create recolours for their objects, but don't yet understand how to use the textures from one object across multiple objects. If you don’t know what I’m talking about when I say “use the object workshop” or “in the resource tree”, this tutorial is not for you!

That said, I have tried to keep the explanation simple enough for beginners to try to follow along as well.

For the purposes of this tutorial I’m going to start by using BeOSBoxBoy’s beautifully modelled Futonesque Fantasy Loveseat from TSR. With his permission I recently made and released a repository version of it, so you can download his original object to work through the tutorial with, and my amended version to look at if you want to see the actual changes that were made.

Note – more changes then are described here were made to the version I published. So as not to interfere with his original version, mine was assigned its own unique GUID, internal name, etc. If you're reasonably familiar with making custom objects, the additional changes shouldn’t phase you when comparing the two files.

Part 1 – Gathering Information

The first thing we’re going to need to do is to look up some information from the original Maxis Futonesque Fantasy sofa. There are three key pieces of information we need: the internal name of the object, the names of the parts of the object, and the name of the material applied to each part. Notepad is a great tool to use for keeping track of information like this because we can just copy and paste things back and forth between SimPE and it.
Step 1.1 – Locate Sofa

Using the Object Workshop, locate the Futonesque Fantasy sofa and select it.
Step 1.2 – Open Copy of Sofa

Click on “Next” at the top of the screen. In the next window, select task “clone” and uncheck all the boxes. This tells SimPE that we want to see the unchanged contents of the original object file.

Click “Next” again. There will be a pause and then you’ll get the message that the “Object was created and loaded”. Go and look at the resource tree view of it.
Step 1.3 – Find Internal Name

The first thing we want to look up is the internal name of the object. Click on “Resource Node (CRES)” in the resource tree, and on the entry in the resource list. In the plugin view you’ll see the following:

Where is says “Filename:” is where we find the internal name of the object. Copy and paste everything before the “_cres” – in this case, “sofaValue1” – to your notepad.
Step 1.4 – Find Part Name(s)

Next we need to find out what part or parts have the texture we’re using. Click on “Geometric Data Container (GMDC)” in the resource tree. Look at the resource list. In this case there is just one entry there – some objects might have multiple GMDCs and take a bit more searching then this will. Click on the entry and look at the plugin view. In this case, there are only four parts to the object; two shadow parts and two different “sofa” parts.

If we weren’t sure which was the part (or parts) using the texture(s) we need, we would uncheck everything except the single part we want to check, and then click “Preview” to see if the displayed mesh is indeed using the texture we want.

In your notepad write down the name of the part or parts you want to link. In this case we only have one part name – “sofa”.

---

*TipSims Resource*
Step 1.5 – Find Material Name(s)

Now we need to look up what material is assigned to the part. Click on “Shape (SHPE)” in the resource tree, then on the entry in the resource list (again, some objects might have multiple entries), and look at the plugin view. Click on the “Parts” tab and look for the “sofa” entry in the list.

Here we can see that part “sofa” is assigned the material “sofavalue1_sofa_redchecker”. Write this down. Your notes should now look something like this:

```
sofavalue1
sofa  ->  sofavalue1_sofa_redchecker
```

That ends looking up things from the original mesh. I often at this point also copy down the original item name and description from the “Catalog Description (CTSS)” section, and also check in the “Object Data (OBJD)” on the “Raw” tab to see what kind of price, niceness score, and ratings, etc. the original object has – it’s nice to try and keep these things consistent with a related item.

Close the file. Answer “No” to the “do you want to save” dialog.
Step 2 – Modifying the Custom Object

Step 2.1 – Open the Custom Item

Locate and open the custom package that we’re editing – in this example, that’s the file “beos_valueLoveSeat.package” from wherever we’ve stashed it.
Step 2.2 – Check Part Names

We need to check if the parts of the custom object use the same part names as the original object. They must match for this technique to work. Click on “Geometric Data Container (GMDC)” in the resource tree, then on the entry in the resource list, and look at the plugin view.

Right away we can see that no, the part names aren’t the same. The mesh parts are being called “loveseat” when what we need is “sofa”. Click on the “Groups” tab, and change the two entries. Click “Commit” when you’re done.
**Step 2.3 – Set Material Names**

Next we want to set up the “sofa” part to be looking for the correct material. Go to “Shape (SHPE)”, and click on the entry in the resource list. Go to the “Parts” tab. You’ll see this:

We need to change this to the values from the repository object that we wrote down earlier – part “sofa” using material “sofavalue1_sofa_redchecker”. After the changes are made it will look like this:

Click commit when you’ve made the changes.
Step 3 – Pointing the Custom Mesh at the Repository Mesh

Step 3.1 – Set Up The Part(s) To Be Designable

Click on “Geometric Node (GMND)” in the resource tree and then on the entry in the resource list. Look at your plugin view. There’s a drop-down list named “Blocklist”. In the list, locate the entry named “tsDesignModeEnabled” and select it.

We’ll see a list of what parts are set up to designable. Remember, the game only supports having two “designable” parts to any one mesh, so there should never be more then two entries here! In this case there is one, showing the “loveseat” part again:

Once again, we need to change this to say “sofa” instead. Click commit.
Step 3.2 – Referencing the Repository Mesh

Now we need to set up the package so it knows where to look for the material(s) we used! This part sounds more complex than it actually is, as it involves several steps in sequence. We need to add a blocklist entry, enable it, and set its values. We’ll do it in baby steps. This all takes place where we already are – within the “Geometric Node (GMND)”.

Click on the “Edit Blocks” tab. At the right of the screen there is a dropdown menu. Locate the entry for “(cDataListExtension)” and then click “Add”. You’ll see an unnamed entry appear in the list on the left, like this:

![Image of Edit Blocks tab with (cDataListExtension) added]

You may click “Commit” at this point but it’s optional until the end of this step.

Next we need to enable the entry we just made, so the game knows to use it. Click on the “Content” tab, then on the “cObjectGraphNode” sub-tab. You’ll see this:

![Image of Content tab with cObjectGraphNode selected]
At the far right click where it says “add”. You’ll see an entry appear in Datalist Extension List that read “0: 0x00, 0x00”.

Select this entry, and change the “Enabled” entry to “0x01” and the “Index” number to the next number in hexadecimal sequence from whatever the value of the line number entry before this value is. In this case the previous entry was line “7” so we enter this as 0x00000008, like so:

Again, you may optionally click commit at this point, but it’s not really necessary yet..

Click on the “Blocklist” dropdown at the top, and find the entry we just created and enabled. Select it.

Now we can enter the name of this block, and the value(s) for it. The name we need to use is “tsMaterialsMeshName”. This lets the game know that this information is telling it which mesh objects to look at to find the material(s) we need.
At the far right change the dropdown menu from “Array” to “String”. A blank entry will appear in the list in the middle. Select this, and then at the right set the Name to be our part name - “sofa” - and the String to be the internal name of the object where its texture can be found – the value we wrote down back in Step 1.3, “sofaValue1”. Your screen should now look like this:

Click commit at this point!

We’ve now finished the single most complex step of this whole thing and are almost done.

**Note – For advanced users only!**

If you look at the blocklist, you’ll see there was an entry named “loveseat” of type “cDataListExtension” that no longer has any relevance to the item. Instead of adding a new block entry, we could have just edited the values of this entry – changing its name, deleting the Array entries in the list and putting in our String entry instead. This saves us the steps of adding and enabling a cDataListExtension block since the block is already there. However, you should only do this if you’re positive the entry you’re changing is no longer needed!
Step 4 – Cleaning Up The File

The package still contains a bunch of stuff we don’t use any more. We can now locate and delete all of the following:

1) Under “Texture Image (TXTR)” get rid of the texture showing the surface of the object. Keep the shadow texture though!
2) Under “Material Definition (TXMT)” get rid of all of the materials that aren’t the shadow.
3) Remove all “Material Override (MMAT)” entries.

Save the file – congratulations, you’re done!

Once you’re sure an object you’ve modified this way works correctly in game, don’t forget that you can also eliminate any recolours it used to have, since they are no longer used by the object. Remember when posting repository objects to let users know which object it uses the texture(s) of, so they can figure out what to recolour in order to recolour your object.

Note – a slave object will no longer show up textured when previewing it in the “Geometric Data Container (GMDC)” view, as SimPE doesn’t have the externally referenced texture(s) available. Instead it will display the mesh in white. This is working as intended, not an error with your linking, so don’t panic.
Step 5 – Optional Stuff

At this point you might want to be sure the object description and the settings in the Object Data bear some relationship to the description and settings of the original object. It's nice to keep them relevant to each other, so you don't clone the style of a cheap “no room rating” item and accidentally make the related item be expensive and have a high room score.

I like to keep the price in some relationship to the original as well, usually I just multiply (or divide, as applicable) by some amount. For example these days when I make a set of counter-matching tables, I set the kitchen sized one to be the same cost as the counter is, and then the dining and buffet sizes are 2x and 3x that cost (I wasn't always this methodical, some of my earliest matching items relied on the good old “pull a random number from thin air” technique instead).

Step 6 – Random Notes and Some Advanced Stuff

A few random pointers about this technique:

When using this technique to link custom meshes to other custom meshes, you must must MUST (I cannot emphasize that enough!) remember to include the #0x1C050000! prefix on the internal name and the material name so that the game knows to look for the material in its custom objects section and not among the original Maxis files. Forgetting to append the prefix was always my worst stumbling block when I first learned this technique.

You can have more then one part that pulls its texture from another object. If the material to be used isn't designable, you can even have more then two parts doing this! Useful to take advantage of some of the Maxis predefined textures for making glass or chrome bits.

You can pull textures from more then one repository object by entering more then one string pair for “tsMaterialsMeshName”. For example, my Bentform Change Table gets the texture for its wood and metal from my Bentform Dining Table, and the texture for the mattress pad from the Maxis crib (yes, from the crib not the change table – not everyone makes a retexture of the change table to match their retexture of the crib, bah!).

You cannot pull different materials for parts of the same name! Each part can only have one material. For example, crib bedding and real bedding are both part name “bedding” so you can't have an object that pulls both, since it would require two parts both named “bedding” and the
game would have no way to tell them apart. This is an issue to watch out for when designing objects that get their textures from multiple meshes, as there are a few part names that are reused frequently across objects, such as “sofa”, “wood”, “frame”, “chair”, “bedding”, “fabric”, and so forth. It’s always best to get your part names, material names, and exported meshes/textures to map your new mesh to, ahead of time. I save the resultant text file along with any exported meshes and textures that I’m using as the base for my Maxis-matching creation.

Some times you’ll open a Maxis item to find its internal name and so forth and discover it’s a slave of some other object. In that case, go to its Geometric Node and look at the “tsMaterialsMeshName” entry to get the correct internal name. The part and material names will still be correct.

In Closing…

Huge thank you to the many people here on TSR and on other modding sites (notably Mod The Sims 2) who helped me learn this technique years ago. I hope this explanation and follow-along example of it helps a few more people to learn it too!

Any questions about using this technique can be directed to me here on TSR via private message.