Annotated Object Replacement LScript.

/* @warnings is good for alerting the artist to potential problems within the script. @version sets the minimum version of Lscript to use. Finally, @name is the name of the plug-in which will appear in LightWave3D's plug-in manager.

```
@warnings
@version 2.3
@name ReplaceSelected
```

We set the type of script to a generic class script.

```
generic
{
```

We define the (global) variables that we want the whole script to have access to. If we did not use global variables, portions of the script would not be able to read and set values. In addition, we use recall to retrieve values from previous script executions. Recall's parameters are: the variable you would like to recall followed by the default value of the variable. The default value is only important for the initial execution, after which the script will use the recalled value.

```
    replaceType = recall("replaceType", 1);
    replaceList = recall("replaceList", "");
    replaceMethod = recall("replaceMethod", 1);
    replaceObject = recall("replaceObject", "");
    vColor = recall("vColor", 1);
```

We create an array of the currently selected meshes in the scene that will be replaced and initialize the variables.

```
  objSelection =  Scene().getSelect(MESH);
  objectListSize = 0;
  objectList;
```

This If statement will only continue if the objSelection array is not empty.

```
  if(objSelection)
  {
```

Now we create a dialog to allow the artist to interact with the script. Then we create control variables that will allow us to retrieve the values from the dialog. The necessary controls for our script are: the path to the object list, path to a specific object, an option to choose between the List, Object, or Nulls, an option for either Sequential or Random, and finally an option to change the Color of the objects.

```
      reqbegin("ReplaceSelected");
      c0 = ctlfilename("Replacement List", replaceList, 70, true);
      c1 = ctlfilename("Replacement Object", replaceObject, 70, true);
      c2 = ctlchoice("Replace With",replaceType,@"List","Object","Null"@);
      c3 = ctlchoice("Replace Method",replaceMethod,@"Sequential","Random"@);
      c4 = ctlpopup("Color",vColor,@"- No Change -","Black","Dark Blue","Dark
Green","Dark Cyan","Dark
Red","Purple","Brown","Grey","Blue","Green","Cyan","Red","Magenta","Orange","Whi
te"@);
```

The ctlactive command will be useful to decrease confusion, because it will only enable the controls that match the criteria the artist selects.

```
    ctlactive(c2, "isList", c0);
    ctlactive(c2, "isList", c3);
    ctlactive(c2, "isObject", c1);
```

The following statement tells the script to halt if the user cancels the dialog.

```
    return if !reqpost();
```

Now we retrieve our variables from the dialog and set our variables to these new values.

```
    replaceList = getvalue(c0);
    replaceType = getvalue(c2);
    replaceMethod = getvalue(c3);
    vColor = getvalue(c4);
    replaceObject = getvalue(c1);
```

The store command works in tandem with the recall command by saving all of the data inputted for future executions.

```
    store("replaceList", replaceList);
    store("replaceType", replaceType);
    store("replaceMethod", replaceMethod);
    store("vColor", vColor);
    store("replaceObject", replaceObject);
```

Now we end the dialog.

```
    reqend();
```

This If statement will ensure that we have the replacement list option selected and also that the replacement list path is not blank. Next it will try to read the list line by line and store the information as variable ‘f’. If the file cannot be found, our script will throw an error and halt the script from proceeding. The following If statement will check to make sure that there is indeed at least one line of text in the replacement list before proceeding.

```
  if(replaceType == 1 && replaceList != "")
  {
```

One quirk of LightWave3D is that when it reads files it reads the last line of text as nil. This would pose a problem in our script because the last object in the list would never be used. The following statement corrects this problem by detecting whether LightWave can read the last line properly. If it has trouble, it appends the list with an empty line. If it does not detect a problem, it will just continue on.

```
    f = File(replaceList,"r") || error("Cannot open file: ",replaceList,"");
    f.line(f.linecount());
    if(!f.read())
    {
       f.reopen("a");
```

```
      f.nl();
    }
    f.close();

    f = File(replaceList,"r") || error("Cannot open file: ",replaceList,"");
    if(f.linecount() > 1)
    {
```

The while loop will continue as long as the current line is not the end of the file. The loop will read each line and check to see if it contains .lwo (a LightWave object file). If it does, it will store it to an array that we will use for the replacing object(s).

```
      while(!f.eof())
      {
        curLine = f.read();
        if(curLine.contains(".lwo"))
        {
          objectListSize++;
          objectList[objectListSize] = curLine;
        }
        if(!curLine.contains(".lwo") && curLine != nil)
        {
```

If the user has entered something other than a blank line or an object file, they will be warned.

```
          warn("At least one file in the list is not a .LWO file. File ignored.");
        }
      }
    }
    else
    {
```

If there are no lines in the text file we throw an error.

```
      error("Problem reading file: No files detected.");
    }
  }
```

Below we have an If statement which determines the type of replacement. Since the code is relatively similar to cycle through all of the objects based on their object IDs, we will continue to the more specialized parts.

For the sequential list we use a counter to keep track of the items we have to choose from. Next we check to make certain that as we replace objects we increment our counter to keep track of the item we are replacing with. If we reach the end of the list, we set the counter back to the first object in the list and continue processing.

```
    if(replaceType == 1)
    {
      seqObject = 1;
      for(i = 1; i <= size(objSelection); i++)
      {
        SelectItem(objSelection[i].id);
        if(replaceMethod == 1)
        {
         if(seqObject > size(objectList))
         {
          seqObject = 1;
         }
         ReplaceWithObject(objectList[seqObject]);
         seqObject++;
        }
        else
        {
```
We use the random command to choose a random number between the minimum value (in this case we set it to 1) and our maximum (size of our object list). We can then take this random number and plug it back into our objectList array to retrieve a random object from the list.
```
       randObj = random(1, size(objectList));
       ReplaceWithObject(objectList[randObj]);
      }
      message(i,size(objSelection));
     }
    }
```
For replacing a specific object our code gets even more straightforward. We simply tell LightWave3D what object (path) we want to replace the current object with.
```
    if(replaceType == 2)
    {
     for(i = 1; i <= size(objSelection); i++)
     {
      SelectItem(objSelection[i].id);
      ReplaceWithObject(replaceObject);
      message(i,size(objSelection));
     }
    }
```
This is the Null replacement portion of the script. We utilize the ReplaceWithNull command (using Null for the name) to replace the objects.
```
    if(replaceType == 3)
    {
     for(i = 1; i <= size(objSelection); i++)
     {
      SelectItem(objSelection[i].id);
      ReplaceWithNull("Null");
      message(i,size(objSelection));
     }
    }
    for(i=1; i <= size(objSelection); i++)
    {
```
If the artist chooses to recolour the meshes, this is where we apply those colours. We convert vColor to an integer and subtract by 2 to match LightWave3D's colour order. Then we essentially turn it back into a string and combine it with the LightWave3D command sequence ItemColor. Since ItemColor is not a native LScript command we have to combine all of the parameters of the command together and then feed that string into CommandInput. This allows us to use command sequences inside of LScript.
```
     if(vColor != 1)
     {
      vC = integer(vColor) - 2;
      color = "ItemColor " + vC;
      CommandInput(color);
     }
```
Here we add the other items back to the selection.
```
    AddToSelection(objSelection[i].id);
   }
  }
 if(!objSelection)
 {
```
If there are no selected meshes in the scene, alert the artist and stop the LScript from proceeding.
```
  error("Please select a mesh item and then run the script again.");
 }
}
```

Below are UDFs (User Defined Functions) which process the code independently from the main portion of the script. These will minimize our code by placing repetitive code in a UDF and simply calling it when we need it. We send the value from the dialog to check if the List choice is selected. If so, it will return a true value enabling the proper controls, otherwise it will disable the controls.

```
isList: value
{
 if(value == 1)
  return 1;
 else
  return 0;
}

isObject: value
{
 if(value == 2)
  return 1;
 else
  return 0;
}
```

Here our custom UDF is used for displaying feedback to the user. This function will display the progress of the script at the bottom of the screen in the status bar.

```
message: value, selectionsize
{
 statmessage = string(value) + " of " + string(selectionsize) + " replaced.";
 StatusMsg(statmessage);
}
```