```
/* CHANGING EFFECTS VIA PROPERTY BROWSER AND MESSAGES */
```

```
/* You will need to build a sample asset with these tags, meshes, tracks and triggers
included.  A small engine shed with opening doors.  A flashing light and a name effect
above the doors perhaps and an optional submesh, perhaps a chimney.  You will also
need a texture-group if you plan to implement skin swapping.

Comment out any lines which apply to effects which you haven't implemented yet.  You
should be aware that running SetMeshVisible() or SetFXNameText() when the referenced
effects are not present or are incorrectly configured will crash the game.

The asset is defined as kind buildable but the same principles will apply to almost any
object.
*/
```

CONFIG.TXT

Config.txt file references which will be needed by the sample scripts. This file is for a buildable asset but the entries would be the same for any scriptable kind.

```
kind buildable                                  asset kind
script myClass                                  script file reference
class myClass                                   script class reference
kuid-table {
   skins                <kuid:12345:67890>      kuid of texture-group to be used for skin swapping
}
mesh-table {
   default {
      mesh              default.im

                                                shed door animation to be controlled via messages
                                                received from a vehicle entering the shed and, for test
      anim              anim.kin                purposes, by a link in the Property Browser
      effects {
         corona0 {                              corona to be controlled from the property browser
            kind        corona
            att         a.corona01
            texture-kuid <kuid:123:456>         corona will be illuminated by default using this texture
            object-size 0.5
         }
         name0 {                                name text to be controlled from the property browser
            kind    name
            fontsize    0.5
            fontcolour  255,255,0
```

```
            att             a.name1a

            name
         }
         skin0 {
            kind            texture-replacement
            texture         colour.texture
         }
   }
   submesh0 {

      mesh                 submesh.im

      auto-create          0
   }
}

attached-track {
   track1 {
      track                <kuid:123:4567>
      vertices {
         0                 a.track1a
         1                 a.track1b
      }
   }
}

attached_trigger {
   trigger1 {
      att                  a.trigger1
      radius               50
   }
}
```

name effect will initially be an empty string to be controlled from the property browser

door skin to be controlled from the property browser

initial texture will be as defined in the gmax model

weather vane
hideable submesh to be controlled from the property browser
submesh will be invisible by default until shown by the script

```
/* SCRIPT FILE – SAVE IN THE ASSET FOLDER AS MYCLASS.GS

In simple assets you are not responsible for organising programme flow.  TRS will tell your object what to do and when to do it via a series of predefined method calls.  These are triggered by various events within the game environment.  All that you have to do is to decide which game events you want to respond to and to tell TRS what to do when the corresponding method calls are received.
*/
```

```
include “buildable.gs”                                          List of standard scripts to include

    class myClass isclass Buildable {                           The name of your script class and immediate parent

        Asset skins;                                            global variable to contain a reference to the texture-
                                                                group
        Asset corona;                                           corona texture for script use
        string name = “”;                                       name string for script use
        bool submeshVisible = false;                            boolean variable for submesh visibility
        bool coronaVisible = true;                              boolean variable for corona visibility
        bool doorsOpen = false;                                 boolean variable for door opening
        int skin = 0;                                           global variable to hold the value of the current texture

/* VehicleHandler is a message handler method that you will define to carry out the
required action whenever your object receives an “Object” message from a vehicle.
*/

    void VehicleHandler(Message msg) {
        Vehicle vehicle = cast<Vehicle>msg.src;                 check the source of the message is a vehicle
        if (!vehicle) {return;}                                 if it is not a vehicle then exit the method
        if (msg.minor == "InnerEnter") {doorsOpen = true;}      on InnerEnter set the doorsOpen variable to true
        else if (msg.minor == "InnerLeave") {doorsOpen = false;} on InnerLeave set the doorsOpen variable to false
        SetMeshAnimationState(“default”,doorsOpen);             set the animation to match the doorsOpen variable
    }

/* GetCorona is a user declared utility function which retrieves the corona texture  from
config.txt.  Since the code required to do this is rather cumbersome it is convenient to
declare it as a subroutine.  You can use similar techniques to retrieve information from
any part of the config.txt file for any asset that you can access.
*/

    Asset GetCorona(string mesh, string effect) {
        Soup meshtable = GetAsset().GetConfigSoup().GetNamedSoup(“mesh-table”);     get the asset mesh-table
        Soup effects = meshtable.GetNamedSoup(mesh).GetNamedSoup(“effects”);        get the effects subtable

        KUID kuid = effects.GetNamedSoup(effect).GetNamedTagAsKUID(“texture-kuid”);  get the kuid of the asset specified by the texture-kuid
                                                                                     tag
        return World.FindAsset(kuid);                                                return the asset referenced by the tag
    }

/* Init is called by the Game when your object is first initialised.  Here you should set up
the default state of the object and provide default values for any global data variables
that you plan to use
```

```
*/

    public void Init(void) {
        inherited();                                                    call any code defined by parent objects
        skins = GetAsset().FindAsset(“skins”);                          assign texture-group to variable
        corona = GetCorona(“default”,”corona0”);                        assign corona texture defined in config.txt
        AddHandler(me,"Object","","VehicleHandler");                    listen for Object messages and pass to VehicleHandler
    }

/* SetProperties is called by the game when any data which has been saved to the
session file is to be recalled.  This occurs when the session is first loaded and before the
Object Property browser is called.  This method is also when the user presses UNDO.in
Surveyor.
*/

    public void SetProperties(Soup soup) {
        inherited(soup);                                                call any code defined by parent objects.
        skin = soup.GetNamedTagAsInt(“skin”,skin);                      assign any saved value for skin.
        coronaVisible = soup.GetNamedTagAsBool(“coronaVisible”,coronaVisible);   assign any saved value for corona visibility.
        submeshVisible = soup.GetNamedTagAsBool(“submeshVisible”,submeshVisible); assign any saved value for submesh visibility.
        string temp = soup.GetNamedTag(“name”);                         assign any saved value for the name effect.
        if (temp != “”) {name = temp;}                                  if there is no saved value use the default.

                                                                        SetProperties is the mechanism used by Trainz to
                                                                        implement its UNDO/REDO system.  To ensure that this
                                                                        is kept in step we need to make sure that the state of
                                                                        the object is kept up to date whenever SetProperties is
                                                                        called. Although this will duplicate code we are using to
                                                                        implement real time changes via the browser interface
                                                                        this will only ever happen in Surveyor so we don’t need
                                                                        to worry too much about performance.

        SetFXTextureReplacement(“skin0”,skins,skin);                    assign the skin
        if (coronaVisible) {SetFXCoronaTexture(“corona0”,corona);}      if the corona is visible then turn it on
        else { SetFXCoronaTexture(“corona0”,null);}                     else turn it off
        SetFXNameText(“name0”,name);                                    set the name effect text string
        SetMeshVisible(“submesh”,submeshVisible,0.0);                   set the submesh on or off
    }

/* GetProperties is called by the game when data needs to be saved to the session file.
This occurs when the session is saved by the user and after the Object Property browser
is closed.
```

```
*/

    public Soup GetProperties(void) {
        Soup soup = inherited();                                          call any code defined by parent objects
        soup.SetNamedTag(“skin”,skin);                                    save the current value of skin texture
        soup.SetNamedTag(“coronaVisible”,coronaVisible);                  save corona visibility
        soup.SetNamedTag(“submeshVisible”,submeshVisible);                save submesh visibility
        soup.SetNamedTag(“name”,name);                                    save the name effect text
        return soup;                                                      pass the soup database back to the game
    }

/* GetDescriptionHTML is called by the game when the Object Property browser is about
to be opened or refreshed.  This is where you set out the HTML code that the browser will
display including the links which will be needed to modify your object.
*/

    public string GetDescriptionHTML(void) {
        string html = inherited() + “<br>”;                               retrieve any HTML defined by parent objects
        return html                                                       pass this back to the game with your own code added
        + “Skin: <a href=live://property/skin>” + skin + “</a><br>”       supply a link for the skin
        + “Corona: <a href=live://property/corona>” + coronaVisible + “</a><br>”   supply a link for the corona
        + “Submesh: <a href=live://property/submesh>” + submeshVisible + “</a><br>”   supply a link for the submesh
        + “Doors: <a href=live://property/doors>” + doorsOpen + “</a><br>”   supply a link to test the doors
        + “Name: <a href=live://property/name>” + name + “</a><br>”;      supply a link to edit the name
    }

/* GetPropertyType is called by the game when a link is clicked in the Object Property
browser.  This is where you tell TRS what data type the linked property represents.  The
data type “link” means that the user’s click is all the information that is necessary for the
game to carry out any action required and to update the browser.  Other types, such as
“string” or “list” will call up an edit box allowing the user to provide typed input.
*/

    public string GetPropertyType(string p_propertyID) {
        string result = inherited (p_propertyID);                         retrieve any value set by the parent
        if (p_propertyID == "skin") {result = "link";}                    this property should be treated as a link
        else if (p_propertyID == "corona") {result = "link";}             this property should be treated as a link
        else if (p_propertyID == "submesh") {result = "link";}            this property should be treated as a link
        else if (p_propertyID == "doors") {result = "link";}              this property should be treated as a link
        else if (p_propertyID == "name") {result = "string";}             this property should be treated as a text string
        return result;                                                    send the answer back to the game
    }
```

```
/* LinkPropertyValue is called by the game to find out what action needs to be taken
when a link is clicked in the  Object Property browser.  Here you should change the values
of your global variables to allow the game to update the browser to match the new
values.  If you want the object to change in real time you should also execute the
necessary method calls.
*/

public void LinkPropertyValue(string p_propertyID) {
    if (p_propertyID == “skin”) {                                                  if the user clicked on skin
        skin++;                                                                    add one to its current value
        if (skin >= skins.GetConfigSoup().GetNamedSoup(“textures”).CountTags()) {  if the new value is greater than the skins available
            skin = 0;                                                              set it to the first skin
        }
    }
    else if (p_propertyID == “corona”) {                                           else if the user clicked on corona
        coronaVisible = !coronaVisible;                                            reverse the value of coronaVisible
        if (coronaVisible) {SetFXCoronaTexture(“corona0”,corona);}                 if the new value is on then turn the corona on
        else { SetFXCoronaTexture(“corona0”,null);}                                else turn it off
    }
    else if (p_propertyID == “submesh”) {                                          else if the user clicked on submesh
        submeshVisible = !submeshVisible;                                          reverse the value of submeshVisible
        SetMeshVisible(“submesh”,submeshVisible,0.0);                              set the submesh visibility to the new value
    }
    else if (p_propertyID == “doors”) {                                            else if the user clicked on Doors
        doorsOpen = !doorsOpen;                                                    reverse the value of doorsOpen
        SetMeshAnimationState(“default”,doorsOpen);                                set the animation to suit
    }
    else inherited(p_propertyID);                                                  else call the parent object
}

/* GetPropertyName is called by the game to find out what title should be provided for
the edit box when a property which requires typed input is called.  If all your properties
are of type “link” you will not need to implement this method.
*/

public string GetPropertyName(string p_propertyID) {
    string result = inherited(p_propertyID);                                       check with the parent object
    if (p_propertyID == “name”) {result = “Enter Name Text”;}                      if the property is ‘name’ use this caption
    return result;                                                                 send the result to the game
}
```

```
/* GetPropertyValue is called by the game to retrieve the current value of the linked
property.  There are various versions of this method to cover the different data types.  If
all your properties are of type “link” you will not need to implement this method.
*/

    public string GetPropertyValue(string p_propertyID) {
        string result = inherited(p_propertyID);                  check with the parent object
        if (p_propertyID == “name”) {result = name;}              if the property is ‘name’ then get the current text
        return result;                                            send the result to the game
    }

/* SetPropertyValue is called by the game to set the new value of the linked property.
There are various versions of this method to cover the different data types.  If all your
properties are of type “link” you will not need to implement this method.
*/

    public void SetPropertyValue(string p_propertyID, string value) {
        if (p_propertyID == “name”) {name = value;}               if the property is ‘name’ then set the new value
        else inherited(p_propertyID,value);                       else pass the data to the parent class
    }

};                                                                End of script
```