Entities

General

Generals all use .ent files that are created in the model editor tool. Exactly how an entity functions is largely up to the app using the engine. Entities can be background props, enemies, and pretty much whatever. For more specific information on the basic types that are provided in the default game setup, check the Game Scripting Guide.

On this page the more general structure of entities will be covered.

Lowlevel Structure

At the very basic level an entity comes as an .ent file that is loaded by the engine. The engine provides a basic loader for entities that handles loading of the basic types contained in it. The game can provide their own loader built from scratch, but one almost always wants to use the loader that the engine provides. The default game provides a few basic types like like Props (doors, buttons, dynamic crates, etc) and Agents (enemies, NPCs, etc) that are used as a foundation for the actual entity used in the game. The final entity type is defined by a script file, which takes care of all specific entity loading, setup and logic.

In order to specify a type it first needs to be added to **“config/EntityTypes.cfg”** under the basic game type is should used (Prop, Agent, etc). Here the name (used as an identfier), the script file, the class name (refering to the main class in the script file) and if it is forced to have full game save (if all data is always saved) is specified. Then the editor must also know about this type and do do this **“edtitor/EntityClasses.def”** needs to be updated with the type's variables. Once this is done, the type is ready to be used in game and editor.

Example:
The entity “Prop_Lamp” uses the basic type “Prop” and is meant to be used for any lamp-like objects in the game. The file **“config/EntityTypes.cfg”**has been updated with:

```
<PropType
  Name = "Prop_Lamp"
  ScriptFile = "props/Prop_Lamp.hps"
  ScriptClass = "cScrPropLamp"
  ForceFullGameSave = "false"
/>
```

This is added to the `PropTypes` element and contains the needed data for the game
To use it in the editor, **“edtitor/EntityClasses.def”** has also been updated with the following:

```
<Class Name="Prop_Lamp" InheritsFrom="Prop">
    <EditorSetupVars>
        <Var Name="AffectLightsVar" Value="Lit" />
        ...
    </EditorSetupVars>
    <TypeVars>
```

```
        <Var Name="CanBeLitByPlayer" Type="Bool"  DefaultValue="true"
Description="If the player can lit this lamp by direct interaction." />
        ...
    </TypeVars>
    <InstanceVars>
        <Var Name="Lit" Type="Bool" DefaultValue="true" Description="If the
lamp is lit or not." />
        ...
    </InstanceVars>
</Class>
```

This makes the editors (model and level) aware that the type exists and provides them with info on what variables it should have as well as any special behavior. More specifics are found below.

Data Structure

An entity is built up from several different elements, all contained or referenced to by the ent file. None of these are required, and an entity can contain all of these types, just a single one or any variation (except for animations which require a mesh).

Mesh
This is the core object for almost all entities. The data is a mesh file (".dae" or ".msh") that can that comes in the form of a separate file which the entity references to. An entity can only have a single mesh connected to it. Note that several entities can connect to the same mesh file.

Animations
An entity file can have one or more animations. In order to have animations there needs to be a mesh connected to the entity file as well. Animations are connected to the entity as separate files (".dae_anim" or ".anim"). It is okay if several entities share the same animations.

Bodies
This gives the entity its physics properties and allows it to collide and interact with the game's world and other entities. A body is made up from more or many shapes. If the mesh has several submeshes and there are several bodies, it must be specified which submesh belong to which body. This data is defined in the entity file.

Joints
These are used to joint up several bodies in various ways (e.g. create a door hinge between door and frame) or just attach a single body to the world in some way. This data is defined in the entity file.

Effects
This includes stuff like particles, billboards, flares, sounds, etc. Everything that can be added to spice the entity up. If there are multiple bodies then it needs to be specified where each effect is attached. Effects can also be attached to the bones of a skinned character. This data is defined in the entity file.

Creation Workflow

Here is summary of the basic kind of worlflow that is used when creating an entity.

The creation of an entity mostly starts out with the creation of a mesh. This mesh is made in a program like Blender or Maya and it is then exported as a Collada file. Matierals then need be used for the mesh so it can be loaded properly into the engine. For the basic steps on how to create a model, check this tutorial.

Oncee the mesh is done the entity file is created. Usually, there is a one-on-one relationship between the ent file and the mesh file (.dae) so it is best to put both in the same folder and let them have the same name. For example, if the mesh has the name "my_model.dae" the entity file should be name "my_model.ent".

It is now time to open the ModelEditor tool.

Next up comes building the physics for the entity. Not all entities require this (for instance Agents), but most do. One or several bodies are then created that approximate the mesh shape somewhat (here is a guide on how to do that properly). Properties such as mass, material, etc are then set to the bodies and if needed they are connected using joints.

If needed, now is a good time to add any animation files are needed for the model. Usually these are placed in a folder named **"animation"** residing in the same path as the mesh file (if animations are shared, another setup is probably required).

Particles, lights, sounds, etc are now added and attached to bodies or joints if applicable. The entity can be tested inside the model editor to make sure every thing works according to plan. The entity file is saved and the model editor can be closed down.

Now the final step in the model editor: setting up type variables. This is some of the of the variables that have been specified in the "EntityClasses.def" file used by the level editor. Usually this deals with type specific gameplay related properties like how long a door takes to open, how fast a creature is, etc. Each different entity type has a different set of type variables (but share many with other entity types).

Now the entity is ready to be added to the game. To do this the editor is opened and the entity mode is selected. One can now find the menu according to its placement in the file structure (this is usually "entities/[category]/[subcategory]/[entityname]/") and select it. The entity can now be placed in the level. After this, there are some instance variables available (e.g. lamp color, if a door is locked, etc). These variables have also been specified in "EntityTypes.def".

Once the game loads up the saved map file the entity will appear there, with bodies, effects and everything.

Level Editor variables

The level editor variables are set in the "EntityClasses.def" file. They contain a number of entity types (element Class) that can inherit variables from the elements BaseClass.

The editor contains three different sets of variables that can be defined in each Class element:

EditorSetupVars
These are variables that are used by the editor only. For instance AffectLightsVar specifies that an Instance variable (see below) determines if the light in the entity is shown or not. These are mostly used to make sure user variables affecting the look of the game also show up in the editor.

TypeVars

These are variables that are set on an entity file basis. This is variables that tweaks the behavior and look of the entity depending on the gameplay needs (a crank is faster or slower to turn) or depending on how the data in the entity file (more health given to larger ogres). These variables are changed in the ModelEditor.

InstanceVars

These are set on a per placed entity basis. So each instance of an entity placed in a level can have different values on these properties. This can be things like the current health level, callback functions, connections to other entities, etc. These variables are changed in the LevelEditor.

From:
https://wiki.frictionalgames.com/ - **Frictional Game Wiki**

Permanent link:
https://wiki.frictionalgames.com/hpl3/engine/entities?rev=1351073926

Last update: **2012/10/24 11:18**