Eden properties for an Arma 3 plane

With the advent of Eden customisation of any vehicle or unit can be done basically within the 3d editor before mission start. As we recently got permission to work on the F/A-18 and SU-35 from John_Spartan and Saul, we've added a basic property to these planes for loadout, paint scheme and folding wings.
Before we dive into the details of property definitions and scripts to use, let's be clear that the inspiration and techniques for this come by most part from an existing mod: The ground breaking 3den enhanced mod from Revo.

Overview

When a player wants to customise a F/A-18 or SU-35 before mission start he already has several possibilities. The primary resource should be the init module for the planes. Scripting comes as a close second, making it easier to mass apply changes to several planes at once. However, with the Eden 3d editor these changes are not easily perceived by the player - the base model in its default configuration stay as is.
Here comes the power of the scriptable and moddable Eden editor in play. With the help of attributes which are attached to objects within Eden, custom properties can be defined. The configuration of the object can then be changed via scripts before mission starts. The changes a player makes hence are reflected in the altered object with Eden and later after mission start.

Foldable Wings

The F/A-18 comes with foldable wings. A call to the *animate* command on a specific selection in SQF causes the wings to fold or unfold. The baseline code for this is

```
_this animate ['l_wingfold',1, true];
_this animate ['r_wingfold',1, true];
```

Basically it uses the object *_this* and calls animate instantly on the selections *l_wingfold* and *r_wingfold*. This folds the wings, the opposite effect is achieved by replacing 1 with 0. *True* achieves an immediate effect - no animation will be played. With help of this very simple command we can now define an Eden attribute that lets us fold the wings of the plane:

```
class Attributes {
  class Wings {
    displayName = "Fold wings";
    tooltip = "Fold the wings of the plane";
    property = "ttt_foldWings";
   control = "Checkbox";
    expression = "_this setVariable ['%s', _value]; if (_value) then { _this animate ['l_wingfold',1,
true]; _this animate ['r_wingfold',1, true]; } else { _this animate ['l_wingfold',0, true]; _this
animate ['r_wingfold',0, true]; };";
    defaultValue = false;
  };
};
```

The *attributes* are placed in the *CfgVehicles* section of the plane.
The *displayName* and *tooltip* attributes are self explanatory, they control the appearance of the attribute in Eden itself. The *property* is similar to a variable that holds the value for the attribute. With *control* the attribute type is defined, here we have *true* or *false* available through a checkbox.
The expression contains the beef of the attribute: What to do when the value is altered by the player in the editor? Here we first store the current *_value* in a variable with the attribute property name. Then we check if *_value* is *true* and fold the wings, or unfold them when *false*.
Lastly the *defaultValue* is set to *false*, e.g. the attribute is not set by default.

Loadout

A typical modern jet plane has a multitude of loadout options for its weaponry. While most Arma 3 planes come with sophisticated loadout menus in game, a scenario designer might be more interested in assigning realistic loadout schemes to the planes in a mission. For the purpose of this document we will restrict these loadouts to three: CAP (air to air), CAS (air to ground), Multi-Role.
One way to implement this loadout selection in Eden is with the help of a Combo box that displays several options in a dropdown list. Again we start with the attributes definition and explore the needed infrastructure next.

```
class Loadout {
  displayName = "Choose loadout";
  tooltip = "Select a predefined loadout";
  property = "ttt_loadout";
  control = "TTT_LoadoutCombo";
  expression = "_this setVariable ['%s', _value]; [_this, _value] execVM
'js_jc_fa18\scripts\LOADOUTS\FA18_roles.sqf';";
  defaultValue = "mr";
  typeName = "STRING";
};
```

Here we utilise a custom control, *TTT_LoadoutCombo*. The expression again holds the core of the attribute: when the value is changed, it is stored in *ttt_loadout* in the plane and the FA18_roles.sqf script is executed. The script contains code that actually changes the loadout, with the help of *addWeapon*.
The custom control is defined in *CfgEden* in the config.cpp, in a separate section:

```
#include "\a3\3DEN\UI\macros.inc"

class ctrlCombo; // external
class Cfg3DEN
{
  class Attributes
  {
    class Default;
    class Title: Default
    {
       class Controls {
         class Title;
       };
    };
    class TTT_LoadoutCombo: Title
    {
      attributeLoad = "[_this controlsGroupCtrl 100, _config] call
JS_JC_fnc_FA18_Eden_attributeLoadCombo;";
      attributeSave = "[_this controlsGroupCtrl 100, _config] call
JS_JC_fnc_FA18_Eden_attributeSaveCombo;";

      class Controls: Controls
      {
         class Title: Title{};
         class Value: ctrlCombo
         {
           idc = 100;
           x = ATTRIBUTE_TITLE_W * 2 * GRID_W;
           w = ATTRIBUTE_CONTENT_W * GRID_W;
           h = SIZE_M * GRID_H * 1.2;

           class Items
           {
              class CAP
              {
                 text = "CAP";
                 data = "cap";
               };
               class CAS
               {
                 text = "CAS";
                 data = "cas";
                 };
                 class MR
                 {
                   text = "Multi-Role";
                   data = "mr";
                  };
               };
           };
         };
      };
```

```
   };
};
```

Anyone that ever toyed around with GUIs in arma will see a close familiarity here. From inside to outside, we see that a bunch of classes are defined within Items. These classes hold the display *text* and the *data*. The *data* will be consumed by the attribute as *_value*. *idc*, *x*, *w*, and *h* define the graphical interface for the custom control. The custom control is based on *ctrlCombo* from BI. The *attributeLoad* and *attributeSave* functions are called whenever the GUI is being loaded or saved. Thank to Revo for allowing us to use his functions there, which are derived from BI samples:

```
_ctrl = _this select 0;
_config = _this select 1;

_attCtrl = getText (_config >> 'control');
_staticItemsCfg = configFile >> 'Cfg3DEN' >> 'Attributes' >> _attCtrl >> 'Controls' >> 'Value' >>
'items';

_fnc_setValues =
{
   private ['_index'];
   params ['_path',['_apply',true]];
   {
      _cfg = _x;
      if (_apply) then
      {
         _index = _ctrl lbAdd getText (_cfg >> 'text');
         _ctrl lbSetData [_index, getText ( _cfg >> 'data')];
      }
      else
      {
         _index = _foreachindex;
      };
      if !(_value isEqualType '') then
      {
         if (_index isEqualTo _value) then
         {
            _ctrl lbSetCurSel _index;
         };
      }
      else
      {
         if (_value == getText (_cfg >> 'data')) then
         {
            _ctrl lbSetCurSel _index;
         };
      };
   } forEach configProperties [_path,'isclass _x'];
};
if (isClass _staticItemsCfg) then
{
   [_staticItemsCfg,false] call _fnc_setValues;
};
```

The load function checks the *items* class inside of our configured custom control and loads them into a listbox. If you look closely you see the correlation between *text* and *data* in the script.

```
_ctrl = _this select 0;
_config = _this select 1;

_value = _ctrl lbData lbCurSel _ctrl;

if ((count get3DENselected 'object') > 0) then
{
	_att = getText (_config >> 'property');
	collect3DENHistory
	{
		{
			_x set3DENAttribute [_att,_value];
		} forEach (get3DENSelected 'object');
	};
};

_value
```

The save script saves the attribute in Eden to the object.

Conclusion

The definition of new attributes for object inside of Eden is a powerful method for mod makers to support players. Long gone are the days where cryptic init line scripts had to be used to customize an object. In this article we have shown how to add a simple wing folding and loadout attribute to a plane. Of course this can be adjusted to your own liking. Likewise as we did with the existing sample documentation from BI and Revo's mod.