There is more information in the Sandbox Manual on how to use the Flow Graph Editor.

It's also possible to add Flowgraph functionality to any Lua entity by Adding Lua Flownode Inputs and Outputs.

Overview

This article describes the steps to implement a new flow graph node.

Basic Code

It is recommended to implement flow nodes that belong to the same group in a single cpp files. Usually there is no need for a header except for some special nodes.

Place the cpp file in the project file in `CryAction/Flow System Files/Nodes` and within the file system directory structure in `Code/CryEngine /CryAction/FlowSystem/Nodes`.

In the cpp file, add a class using the following code template and replace "MyName" with the desired name of your flow node. Replace "FlowNodeGroup" with your desired group name and create a corresponding sub-folder in the editor node selector where this node type will be placed.

```
#include "StdAfx.h"
#include "FlowBaseNode.h"

class CFlowNode_MyName : public CFlowBaseNode<eNCT_Instanced>
{
public:
  CFlowNode_MyName(SActivationInfo* pActInfo)
  {
  };

  virtual IFlowNodePtr Clone(SActivationInfo *pActInfo)
  {
        return new CFlowNode_MyName(pActInfo);
  };

  virtual void GetMemoryUsage(ICrySizer* s) const
  {
     s->Add(*this);
  }

  virtual void GetConfiguration(SFlowNodeConfig& config)
  {
    static const SInputPortConfig in_config[] = {
      {0}
    };
    static const SOutputPortConfig out_config[] = {
      {0}
    };
    config.sDescription = _HELP( "A description of this flow node" );
    config.pInputPorts = in_config;
    config.pOutputPorts = out_config;
    config.SetCategory(EFLN_APPROVED);
  }

  virtual void ProcessEvent(EFlowEvent event, SActivationInfo* pActInfo)
  {
     switch (event)
     {
     };
  }
};

REGISTER_FLOW_NODE("FlowNodeGroup:MyName", CFlowNode_MyName);
```

eNCT_Instanced vs eNCT_Singleton

When you declare a FlowNode in code you can choose between Instanced and Singleton in the template parameter of the parent.

If you have a singleton node, it will only create 1 instance of it. you can still have multiple nodes in your Flowgraph, but the memory footprint will be much smaller.

As a general rule of thumb, use singleton whenever you are not keeping any state data (member variables etc.) and prefer singleton over instanced.

Adding Output Ports

Flow nodes have both input and output ports. An output port can be added by altering the *GetConfiguration* function, as can be seen in the following example:

```
class CFlowNode_MyName : public CFlowBaseNode<eNCT_Instanced>
{
public:
  // ...

  virtual void GetConfiguration( SFlowNodeConfig& config )
  {
    static const SInputPortConfig in_config[] = {
      {0}
    };
    static const SOutputPortConfig out_config[] = {
      OutputPortConfig<int>("alertness", _HELP("useful help text")),
      {0}
    };
    config.sDescription = _HELP( "A description of this flow node" );
    config.pInputPorts = in_config;
    config.pOutputPorts = out_config;
    config.nFlags = 0;
  }

  // ...
};
```

OutputPortConfig is a templatized helper function that is useful for filling a small structure with appropriate data.

Various types can be specified but in the sample the type of the port is "int". The available types are: **SFlowSystemVoid**, **int**, **float**, **EntityId**, **Vec3**, **string**, and **bool**. *SFlowSystemVoid* is a special type which represents "no value". *OutputPortConfig* takes three parameters: the name of the output port, some descriptive help text that is shown in the editor GUI and an optional human readable port name which is visible to designers. Make sure that you choose a good port name as changing it later will break all flow graphs that use this node. If you need to change the displayed name, use the third parameter.

To emit a value from the port, you can use the function *CFlowBaseNode::ActivateOutput(pActInfo, nPort, value)*. This function takes a *pActInfo* (typically passed to *ProcessEvent()*), the port identifier (count starting a zero from the top of out_config), and a value of the same type as the port. So to activate our output "alertness" with 3 we could write:

```
ActivateOutput( pActInfo, 0, 3 );
```

Adding Input Ports

An input port can be added in a similar way as an output port by altering the *GetConfiguration* function:

```
class CFlowNode_MyName : public CFlowBaseNode<eNCT_Instanced>
{
public:
  // ...

  virtual void GetConfiguration( SFlowNodeConfig& config )
  {
    static const SInputPortConfig in_config[] = {
      InputPortConfig<int>("someInput", _HELP("useful help text")),
      {0}
    };
    static const SOutputPortConfig out_config[] = {
      {0}
    };
    config.sDescription = _HELP( "A description of this flow node" );
    config.pInputPorts = in_config;
    config.pOutputPorts = out_config;
    config.nFlags = 0;
  }

  // ...
};
```

InputPortConfig is a templatized helper function similar to *OutputPortConfig*. The same types as for output ports can be specified for the input ports as well.

InputPortConfig takes the following parameters:

- *name* [required] - used internally and for saving the graph. It is again recommended to choose the port name carefully as changing it later will break flow graphs using this node. Important: **Do not use the underscore character '_' in port names** as this was used in previous versions to specify a specialized editor for the port,
- *value* - default value of the port when a new node is created,
- *description* - for the tooltip on mouse hover,
- *humanName* - human readable name for display on the node. Use this to visually override a port name without breaking script compatibility,
- *sUIConfig* - a formatted string specifying how the UI should behave when setting the port value. Here you can choose a specialized widget or tune the allowed range of the input. See 'UI Configuration' for more details.

UI Configuration

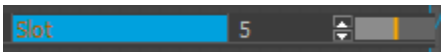
The interface for setting the Input Port value can be defined by passing a series of options in the form of a string with key-value pairs in the *InputPortConfig*.

Possibilities are:

- **Setting the value range:**

```
_UICONFIG("v_min=0, v_max=10")
```

This will limit the widget's arrows and ramp as well as clamp manually inserted values as seen in the picture.



- **Enum dropdown:**

```
_UICONFIG("enum_int:Relaxed=0,Alert=1,Combat=2,Crouch=3")
```

There are several types of enums that can be used to display a dropdown list of readable strings which map to a value that is actually used by the node (and persisted when the graph is saved). Enums can be of type int or float as in the example above and shown in the picture.

An enum can also be of type string with or without mapping to another value:

```
_UICONFIG("enum_string:a,b,c")
_UICONFIG("enum_string:DisplayA=a,DisplayB=b,DisplayC=c")
```

Enums can also refer to the Global and dynamic UI Enums defined in InitUIEnums. For instance, 'vehicleLightTypes' is read from Scripts/Entities/Vehicles /Lights/DefaultVehicleLights.xml . Optionally the enum can depend on another port to affect the available selection:

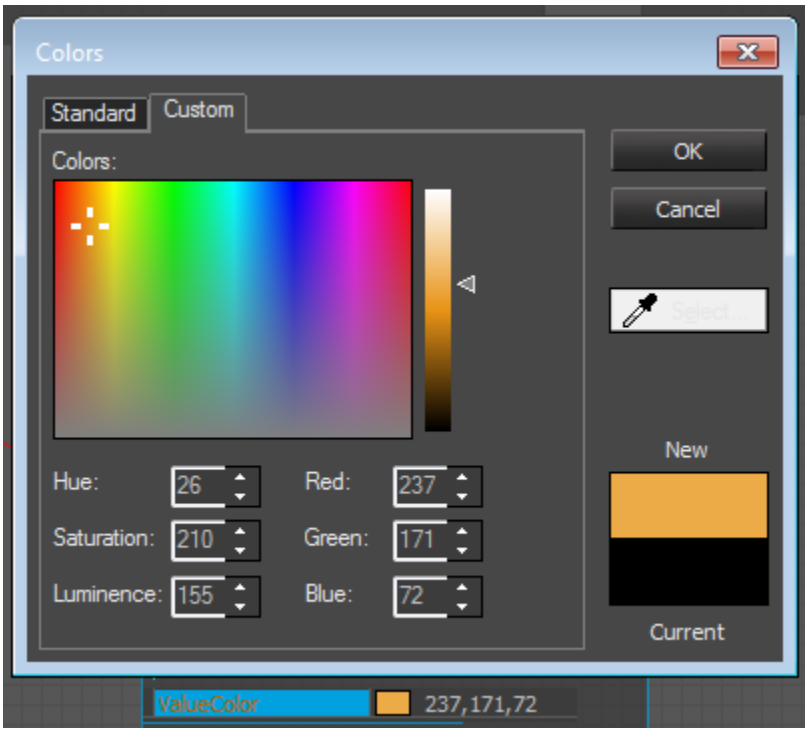
```
_UICONFIG("enum_global:ENUM_NAME")
_UICONFIG("enum_global:vehicleLightTypes")
_UICONFIG("enum_global_def:ENUM_NAME")
_UICONFIG("enum_global_ref:ENUM_NAME_FORMAT_STRING:REF_PORT")
```

- **Specialized Editor**

```
_UICONFIG("dt=editorName")
_UICONFIG("dt=entityProperties, ref_entity=entityId")
_UICONFIG("dt=matparamslot, slot_ref=Slot, sub_ref=SubMtlId, param=float")
```

A dedicated property editor can be indicated with the keyword 'dt', followed by parameters optionally needed by the editor as can be seen in the example above.

There is a set of available editors which name can be consulted in the table below. The following picture illustrates the color picker.



Full engine licensees can add new editors. For that, see the file Sandbox/Editor/HyperGraph/FlowGraphVariables.cpp.

Note: previously the editor names were passed as a prefix to the port name, however this is not recommended because changes to the port name will break compatibility with saved graphs.

Editor Name	Property Editor Type
snd	IVariable::DT_SOUND
sound	IVariable::DT_SOUND
clr	IVariable::DT_COLOR
color	IVariable::DT_COLOR
tex	IVariable::DT_TEXTURE
texture	IVariable::DT_TEXTURE
obj	IVariable::DT_OBJECT
object	IVariable::DT_OBJECT
file	IVariable::DT_FILE
text	IVariable::DT_LOCAL_STRING
equip	IVariable::DT_EQUIP
reverbpreset	IVariable::DT_REVERBPRESET
aianchor	IVariable::DT_AI_ANCHOR
aibehavior	IVariable::DT_AI_BEHAVIOR
aicharacter	IVariable::DT_AI_CHARACTER
aipfpropertieslist	IVariable::DT_AI_PFPROPERTIESLIST
aientityclasses	IVariable::DT_AIENTITYCLASSES
aiterritory	IVariable::DT_AITERRITORY
aiwave	IVariable::DT_AIWAVE

soclass	IVariable::DT_SOCLASS
soclasses	IVariable::DT_SOCLASSES
sostate	IVariable::DT_SOSTATE
sostates	IVariable::DT_SOSTATES
sopattern	IVariable::DT_SOSTATEPATTERN
soaction	IVariable::DT_SOACTION
sohelper	IVariable::DT_SOHELPER
sonavhelper	IVariable::DT_SONAVHELPER
soanimhelper	IVariable::DT_SOANIMHELPER
soevent	IVariable::DT_SOEVENT
customaction	IVariable::DT_CUSTOMACTION
gametoken	IVariable::DT_GAMETOKEN
mat	IVariable::DT_MATERIAL
seq	IVariable::DT_SEQUENCE
mission	IVariable::DT_MISSIONOBJ
anim	IVariable::DT_USERITEMCB
animstate	IVariable::DT_USERITEMCB
animstateEx	IVariable::DT_USERITEMCB
bone	IVariable::DT_USERITEMCB
attachment	IVariable::DT_USERITEMCB
dialog	IVariable::DT_USERITEMCB
matparamslot	IVariable::DT_USERITEMCB
matparamname	IVariable::DT_USERITEMCB
matparamcharatt	IVariable::DT_USERITEMCB
seqid	IVariable::DT_SEQUENCE_ID
lightanimation	IVariable::DT_LIGHT_ANIMATION
formation	IVariable::DT_USERITEMCB
communicationVariable	IVariable::DT_USERITEMCB
uiElements	IVariable::DT_USERITEMCB
uiActions	IVariable::DT_USERITEMCB
uiVariables	IVariable::DT_USERITEMCB
uiArrays	IVariable::DT_USERITEMCB
uiMovieclips	IVariable::DT_USERITEMCB
uiVariablesTmpl	IVariable::DT_USERITEMCB
uiArraysTmpl	IVariable::DT_USERITEMCB
uiMovieclipsTmpl	IVariable::DT_USERITEMCB
uiTemplates	IVariable::DT_USERITEMCB
vehicleParts	IVariable::DT_USERITEMCB
vehicleSeatViews	IVariable::DT_USERITEMCB
entityProperties	IVariable::DT_USERITEMCB
actionFilter	IVariable::DT_USERITEMCB

actionMaps	IVariable::DT_USERITEMCB
actionMapActions	IVariable::DT_USERITEMCB
geomcache	IVariable::DT_GEOM_CACHE
audioTrigger	IVariable::DT_AUDIO_TRIGGER
audioSwitch	IVariable::DT_AUDIO_SWITCH
audioSwitchState	IVariable::DT_AUDIO_SWITCH_STATE
audioRTPC	IVariable::DT_AUDIO_RTPC
audioEnvironment	IVariable::DT_AUDIO_ENVIRONMENT
audioPreloadRequest	IVariable::DT_AUDIO_PRELOAD_REQUEST
dynamicResponseSignal	IVariable::DT_DYNAMIC_RESPONSE_SIGNAL

Trigger Ports

Sometimes it is useful to have a trigger signal as input or output port. Such ports should be implemented using the type *In/OutputPortConfig_Void* or *In/OutputPortConfig_AnyType* and not *bool*.

Update event

Sometimes you want to have an update loop for your node instead of just reacting on ports, below is explained how to do this. You can also only temporarily enable the update event.

The following code will add your node to the list of regularly updated nodes:

```
pActInfo->pGraph->SetRegularlyUpdated( pActInfo->myID, true);
```

This means from now on you will get ProcessEvent calls with eFE_Update event.

To be removed again from this list call the same function with false as the second parameter.

Frequency: you will get 1 ProcessEvent(eFE_Updated) call per Game update call.