Plugins

Plugin architecture

- C++ API (basically the whole code base)
- Reference to shared library (SDF, command line or *gui.ini*)
- Environment variable *GAZEBO_PLUGIN_PATH*
- Subclass a plugin class and override the *Load* method
- Shared pointers, events and messages

Plugin architecture

- C++ API (basically the whole code base)
- Reference to shared library (SDF, command line or *gui.ini*)
- Environment variable *GAZEBO_PLUGIN_PATH*
- Subclass a plugin class and override the *Load* method
- Shared pointers, events and messages

Plugin architecture

- C++ API (basically the whole code base)
- Reference to shared library (SDF, command line or *gui.ini*)
- Environment variable *GAZEBO_PLUGIN_PATH*
- Subclass a plugin class and override the *Load* method
- Shared pointers, events and messages

Plugin architecture

- C++ API (basically the whole code base)
- Reference to shared library (SDF, command line or *gui.ini*)
- Environment variable *GAZEBO_PLUGIN_PATH*
- Subclass a plugin class and override the *Load* method
- Shared pointers, events and messages

Plugin architecture

- C++ API (basically the whole code base)
- Reference to shared library (SDF, command line or *gui.ini*)
- Environment variable *GAZEBO_PLUGIN_PATH*
- Subclass a plugin class and override the *Load* method
- Shared pointers, events and messages

Model plugin base class

```
class GZ_COMMON_VISIBLE ModelPlugin : public PluginT<ModelPlugin>
{
   public: ModelPlugin() {this->type = MODEL_PLUGIN;}
   public: virtual ~ModelPlugin() {}
   public: virtual void Load(physics::ModelPtr _model,
                             sdf::ElementPtr _sdf) = 0;
   public: virtual void Init() {}
   public: virtual void Reset() {}
};
```

Model plugin registration

```
GZ_REGISTER_MODEL_PLUGIN(classname)
```

Plugin types

- World
- Model
- Sensor
- Visual
- System
- GUI

Model plugin

```
GZ_REGISTER_MODEL_PLUGIN(AnimateModel)

class AnimateModel : public ModelPlugin
{
  public: void Load(physics::ModelPtr _model,
                    sdf::ElementPtr _sdf)
  {
    // create the animation
    gazebo::common::PoseAnimationPtr anim(
        new gazebo::common::PoseAnimation("test", 20.0, true));

    // create a keyframe for each pose from input
    gazebo::common::PoseKeyFrame *key;
    int time = 0;
    sdf::ElementPtr poseElem = _sdf->GetElement("pose");
    while (poseElem)
    {
      ignition::math::Pose3d pose =
         poseElem->Get<ignition::math::Pose3d>();
      key = anim->CreateKeyFrame(time);
      key->Translation(pose.Pos());
      key->Rotation(pose.Rot());
      time += 5;
      poseElem = poseElem->GetNextElement("pose");
    }

    // set the animation
    _model->SetAnimation(anim);
  }
};
```

GUI plugin

```
GZ_REGISTER_GUI_PLUGIN(CameraPosesPlugin)

CameraPosesPlugin::CameraPosesPlugin() : GUIPlugin()
    { /* Qt elements */}

void CameraPosesPlugin::Load(sdf::ElementPtr _sdf)
{
  // Fill poses vector
  { // ...
    this->poses.push_back(
        _sdf->GetElement("pose")->Get<ignition::math::Pose3d>());
  }

  // Keep pointer to the user camera
  this->camera = gui::get_active_camera();

  // Filter keyboard events
  gui::KeyEventHandler::Instance()->AddPressFilter(
      "camera_poses_plugin",
      boost::bind(&CameraPosesPlugin::OnKeyPress, this, _1));
}

bool CameraPosesPlugin::OnKeyPress(const common::KeyEvent &_event)
{
  if (_event.key == Qt::Key_Right)
  {  // ...
    this->camera->MoveToPosition(
        this->poses[this->currentIndex], 1);
  }
}
```

System plugin

```
public: virtual void Load(int _argc = 0, char **_argv = NULL) = 0;
```

Thanks for watching! Get involved!

ROS:
http://wiki.ros.org
http://ros2.org
http://design.ros2.org/

Gazebo:
http://gazebosim.org

Run the presentation!
https://bitbucket.org/chapulina/cppcon

Contact us:
Speakers: jackie@osrfoundation.org
louise@osrfoundation.org
ROS 2 mailing list: ros-sig-ng-ros@googlegroups.com
Gazebo mailing list: gazebo@osrfoundation.org