Plugin architecture

- C++ API (basically the whole code base)
- Reference to shared object (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 object (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 object (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 object (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 object (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() {}
};
```


src

Model plugin registration

```
#define GZ_REGISTER_MODEL_PLUGIN(classname) \
  extern "C" GZ_COMMON_VISIBLE gazebo::ModelPlugin *RegisterPlugin(); \
  gazebo::ModelPlugin *RegisterPlugin() \
  {\
    return new classname();\
  }
```


src

Plugin types

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

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)
{
  // Next slide
  if (_event.key == Qt::Key_Right)
  { (...)
    this->camera->MoveToPosition(this->poses[this->currentIndex], 1);
  }
}
```

World plugin

Model plugin

System plugin

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

Gazebo: Conclusion

Get involved:
http://gazebosim.org

Contact:
Speaker: louise@osrfoundation.org
Gazebo mailing list: gazebo@osrfoundation.org