Plugins

"Plugins allow you to control models, sensors, world properties, and even the way Gazebo runs"

★ Plugin 101

⇒ A plugin is a chunk of code that is compiled as a shared library and inserted into the simulation.

⇒ The plugin has direct access to all the functionality of Gazebo through the standard C++ classes.

⇒ You should use a plugin when you want to programmatically alter a simulation.
Ex: move models, respond to events, insert new models etc.

● Plugin Types

⇒ There are currently 6 types of plugins

1. World
2. Model
3. Sensor
4. System
5. Visual
6. GUI

⇒ A plugin type should be chosen based on the desired functionality.

● Hello World Plugin!

⇒ Make sure gazebo developments files are available:

sudo apt-get install libgazebo6-dev

hello_world.cc

```
#include <gazebo/gazebo.hh>

namespace gazebo
{
  class WorldPluginTutorial : public WorldPlugin
  {
    public: WorldPluginTutorial() : WorldPlugin()
            {
              printf("Hello World!\n");
            }

    public: void Load(physics::WorldPtr _world, sdf::ElementPtr _sdf)
            {
            }
  };
  GZ_REGISTER_WORLD_PLUGIN(WorldPluginTutorial)
}
```

All plugins must be in the gazebo namespace.

Each plugin must inherit from a plugin type, which in this case is the WorldPlugin class.

The only other mandatory function is Load which receives an SDF element that contains the elements and attributes specified in loaded SDF file.

Finally, the plugin must be registered with the simulator using the GZ_REGISTER_WORLD_PLUGIN macro.

↳ The only parameter to this macro is the name of the plugin class.

⇒ There are matching register macros for each plugin type:

GZ_REGISTER_MODEL_PLUGIN

GZ_REGISTER_MODEL_PLUGIN

GZ_REGISTER_GUI_PLUGIN

GZ_REGISTER_SYSTEM_PLUGIN

GZ_REGISTER_VISUAL_PLUGIN

⇒ The gazebo/gazebo.hh file includes a core set of basic gazebo functions.

- It doesn't include
  - gazebo/physics/physics.hh
  - gazebo/rendering/rendering.hh
  - gazebo/sensors/sensors.hh
- as those should be included on a case by case basis.

Compiling the Plugin

⇒ To compile the above plugin, create CMakeLists.txt.

```
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)

find_package(gazebo REQUIRED)
include_directories(${GAZEBO_INCLUDE_DIRS})
link_directories(${GAZEBO_LIBRARY_DIRS})
list(APPEND CMAKE_CXX_FLAGS "${GAZEBO_CXX_FLAGS}")

add_library(hello_world SHARED hello_world.cc)
target_link_libraries(hello_world ${GAZEBO_LIBRARIES})
```

⇒ Compiling will result in a shared library, libhello_world.so.

⇒ Lastly, add your library path to the GAZEBO_PLUGIN_PATH:

```
$ export GAZEBO_PLUGIN_PATH=${GAZEBO_PLUGIN_PATH}:~/gazebo_plugin_tutorial/build
```

Using a Plugin

⇒ Once you have a plugin compiled as a shared library (see above), you can attach it to a world or model in an SDF file.

- On startup, Gazebo parses the SDF file, locates the plugin, and loads the code.

★ Model Plugin

⇒ Plugins allow complete access to the physical properties of models and their underlying elements (links, joints, collision objects).

```
#include <functional>
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/common/common.hh>
#include <ignition/math/Vector3.hh>

namespace gazebo
{
  class ModelPush : public ModelPlugin
  {
    public: void Load(physics::ModelPtr _parent, sdf::ElementPtr /*_sdf*/)
    {
      // Store the pointer to the model
      this->model = _parent;

      // Listen to the update event. This event is broadcast every
      // simulation iteration.
      this->updateConnection = event::Events::ConnectWorldUpdateBegin(
          std::bind(&ModelPush::OnUpdate, this));
    }
```

```
    // Called by the world update start event
    public: void OnUpdate()
    {
      // Apply a small linear velocity to the model.
      this->model->SetLinearVel(ignition::math::Vector3d(.3, 0, 0));
    }

    // Pointer to the model
    private: physics::ModelPtr model;

    // Pointer to the update event connection
    private: event::ConnectionPtr updateConnection;
  };

  // Register this plugin with the simulator
  GZ_REGISTER_MODEL_PLUGIN(ModelPush)
}
```

★ World Plugin

```
#include <ignition/math/Pose3.hh>
#include "gazebo/physics/physics.hh"
#include "gazebo/common/common.hh"
#include "gazebo/gazebo.hh"

namespace gazebo
{
class Factory : public WorldPlugin
{
  public: void Load(physics::WorldPtr _parent, sdf::ElementPtr /*_sdf*/)
  {
    // Option 1: Insert model from file via function call.
    // The filename must be in the GAZEBO_MODEL_PATH environment variable.
    _parent->InsertModelFile("model://box");
  }
};

// Register this plugin with the simulator
GZ_REGISTER_WORLD_PLUGIN(Factory)
}
```

★ System Plugin

This tutorial will create a source file that is a system plugin for gzclient designed to save images into the directory /tmp/gazebo_frames.

system_gui.cc

```
#include <functional>
#include <gazebo/gui/GuiIface.hh>
#include <gazebo/rendering/rendering.hh>
#include <gazebo/gazebo.hh>

namespace gazebo
{
  class SystemGUI : public SystemPlugin
  {
    ////////////////////////////////////////////
    /// \brief Destructor
    public: virtual ~SystemGUI()
    {
      this->connections.clear();
      if (this->userCam)
        this->userCam->EnableSaveFrame(false);
      this->userCam.reset();
    }
```

```
  ////////////////////////////////////////////
  /// \brief Called after the plugin has been constructed.
  public: void Load(int /*_argc*/, char ** /*_argv*/)
  {
    this->connections.push_back(
        event::Events::ConnectPreRender(
          std::bind(&SystemGUI::Update, this)));
  }

  ////////////////////////////////////////////
  // \brief Called once after Load
  private: void Init()
  {
  }

  ////////////////////////////////////////////
  /// \brief Called every PreRender event. See the Load function.
  private: void Update()
  {
    if (!this->userCam)
    {
      // Get a pointer to the active user camera
      this->userCam = gui::get_active_camera();

      // Enable saving frames
      this->userCam->EnableSaveFrame(true);

      // Specify the path to save frames into
      this->userCam->SetSaveFramePathname("/tmp/gazebo_frames");
    }
      // Get scene pointer
      rendering::ScenePtr scene = rendering::get_scene();

      // Wait until the scene is initialized.
      if (!scene || !scene->Initialized())
        return;

      // Look for a specific visual by name.
      if (scene->GetVisual("ground_plane"))
        std::cout << "Has ground plane visual\n";
    }

    /// Pointer the user camera.
    private: rendering::UserCameraPtr userCam;

    /// All the event connections.
    private: std::vector<event::ConnectionPtr> connections;
  };

  // Register this plugin with the simulator
  GZ_REGISTER_SYSTEM_PLUGIN(SystemGUI)
}
```