Back (/tutorials?cat=install)

Plugins 101

Overview of Gazebo Plugins

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.

Plugins are useful because they:

- let developers control almost any aspect of Gazebo
- are self-contained routines that are easily shared
- can be inserted and removed from a running system

Previous versions of Gazebo utilized controllers. These behaved in much the same way as plugins, but were statically compiled into Gazebo. Plugins are more flexible, and allow users to pick and choose what functionality to include in their simulations.

You should use a plugin when:

- you want to programmatically alter a simulation

Ex: move models, respond to events, insert new models given a set of preconditions

- you want a fast interface to gazebo, without the overhead of the transport layer

Ex: No serialization and deserialization of messages.

- you have some code that could benefit others and want to share it

Plugin Types

There are currently 6 types of plugins

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

Each plugin type is managed by a different component of Gazebo. For example, a Model plugin is attached to and controls a specific model in Gazebo. Similarly, a World plugin is attached to a world, and a Sensor plugin to a specific sensor. The System plugin is specified on the command line, and loads first during a Gazebo startup. This plugin gives the user control over the startup process.

A plugin type should be chosen based on the desired functionality. Use a World plugin to control world properties, such as the physics engine, ambient lighting, etc. Use a Model plugin to control joints, and state of a model. Use a Sensor plugin to acquire sensor information and control sensor properties.

Hello WorldPlugin!

Plugins are designed to be simple. A bare bones world plugin contains a class with a few member functions.

First, if you installed Gazebo from debians, make sure you've installed the Gazebo development files. If you installed Gazebo from source, you can ignore this step. If you have a release other than gazebo6, replace 6 with whatever version number you have.

```
sudo apt-get install libgazebo6-dev
```

Next, make a directory and a .cc file for the new plugin:

```
$ mkdir ~/gazebo_plugin_tutorial
$ cd ~/gazebo_plugin_tutorial
$ gedit hello_world.cc
```

Copy the following into 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)
}
```

The above code is also located in the Gazebo sources: examples/plugins/hello_world/hello_world.cc (http://bitbucket.org/osrf/gazebo/src/gazebo6/examples/plugins/hello_world), along with an appropriate CMakeLists.txt file.

Code Explained

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

namespace gazebo
{
```

The gazebo/gazebo.hh (https://bitbucket.org/osrf/gazebo/src/gazebo6/gazebo/gazebo_core.hh) file includes a core set of basic gazebo functions. It doesn't include `gazebo/physics/physics.hh`, `gazebo/rendering/rendering.hh`, or `gazebo/sensors/sensors.hh` as those should be included on a case by case basis. All plugins must be in the `gazebo` namespace.

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

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

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

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

```
  GZ_REGISTER_WORLD_PLUGIN(WorldPluginTutorial)
```

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_SENSOR_PLUGIN`, `GZ_REGISTER_GUI_PLUGIN`, `GZ_REGISTER_SYSTEM_PLUGIN` and `GZ_REGISTER_VISUAL_PLUGIN`.

The following section contains instructions on how to compile this plugin.

Compiling the Plugin

Please make sure that gazebo has been properly installed (http://gazebosim.org/install).

To compile the above plugin, create `~/gazebo_plugin_tutorial/` **`CMakeLists`**`.txt`:

```
$ gedit ~/gazebo_plugin_tutorial/CMakeLists.txt
```

Copy the following in 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})
```

New in `gazebo6`: c++11 flags are now required for all downstream software to compile against gazebo. This is done with the following line:

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${GAZEBO_CXX_FLAGS}")
```

Create the build directory

```
$ mkdir ~/gazebo_plugin_tutorial/build
$ cd ~/gazebo_plugin_tutorial/build
```

Compile the code.

```
$ cmake ../
$ make
```

Compiling will result in a shared library, `~/gazebo_plugin_tutorial/ build/libhello_world.so`, that can be inserted in a Gazebo simulation.

Lastly, add your library path to the `GAZEBO_PLUGIN_PATH`:

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

> *Note*: This changes the path only for the current shell. If you want to use your plugin for every new temrinal you open, append the line above to the `~/.bashrc` file.

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 (see SDF documentation (http://gazebosim.org/sdf.html) for more info). On startup, Gazebo parses the SDF file, locates the plugin, and loads the code. It is important that Gazebo is capable of finding the plugin. Either the full path to the plugin is specified, or the plugin exists in one of the paths in the `GAZEBO_PLUGIN_PATH` environment variable.

Create a world file and copy the code below into it. The example world file can also be found in examples/plugins/hello_world/hello.world (https://bitbucket.org/osrf/gazebo/src/gazebo6/examples/plugins/hello_world/hello.world).

```
$ gedit ~/gazebo_plugin_tutorial/hello.world
```

```
<?xml version="1.0"?>
<sdf version="1.4">
  <world name="default">
    <plugin name="hello_world" filename="libhello_world.so"/>
  </world>
</sdf>
```

Now open it with `gzserver` :

```
$ gzserver ~/gazebo_plugin_tutorial/hello.world --verbose
```

You should see output similar to:

```
Gazebo multi-robot simulator, version 6.1.0
Copyright (C) 2012-2015 Open Source Robotics Foundation.
Released under the Apache 2 License.
http://gazebosim.org

[Msg] Waiting for master.
[Msg] Connected to gazebo master @ http://127.0.0.1:11345
[Msg] Publicized address: 172.23.1.52
Hello World!
```