Sigslots

Introduction

Sigslots are the primary way to handle events emitted by the kobuki driver (c.f. with the usual function callbacks with void function pointers as arguments). Rather than repeating here a verbose explanation of what they are and how they work, the best option is to go straight to the official documentation - ecl_sigslots.

Sigslot Reference

The kobuki driver establishes a set of signals on uniquely established namespaces. The namespace for each is separated into two parts - the first is the namespace specified by the sigslots_namespace variable in the kobuki::Parameter strucutre. The second uniquely identifies the signal itself.

The following represent the available signals when namespaced under "/kobuki" (default).

- void : /kobuki/stream_data : informs when a new data packet has arrived from the kobuki
- std::string : /kobuki/ros_debug : relay debug messages
- std::string : /kobuki/ros_info : relay info messages
- std::string : /kobuki/ros_warn : relay warning messages
- std::string : /kobuki/ros_error : relay error messages
- **kobuki::ButtonEvent** : /kobuki/button_event : receive an event when a button state changes
- **kobuki::BumperEvent** : /kobuki/bumper_event : receive an event when the bumper state changes
- **kobuki::CliffEvent** : /kobuki/cliff_event : receive an event when a cliff sensor state changes
- **kobuki::WheelEvent**: /kobuki/wheel_event : receive an event when the wheel state (in/out) changes
- **kobuki::PowerEvent** : /kobuki/power_event : receive an event when the power/charging state changes
- **kobuki::InputEvent** : /kobuki/input_event : receive an event when the gpio state changes
- **kobuki::RobotEvent** : /kobuki/robot_event : receive an event when the robot state changes
- **kobuki::VersionInfo** : /kobuki/version_info : receive version info strings on this signal

It does not establish any slots.

Usage

The */kobuki/stream_data* is the most important slot. The kobuki sends a single long data packet periodically and when this is received, it emits a signal informing you that it has arrived and is ready to be processed. At this point, you can quiz the kobuki driver for the state (**kobuki::Kobuki::getCoreSensorData**) which returns a **kobuki::CoreSensors::Data** structure holding all the important sensor information for kobuki.

Simple Example - Catching Wheel Odometry

A small example program which processes this signal sending current wheel encoder values to standard output:

```
#include <ecl/time.hpp>
#include <ecl/sigslots.hpp>
#include <iostream>
```

```
#include <kobuki_driver/kobuki.hpp>

class KobukiManager {
public:
  KobukiManager() :
      slot_stream_data(&KobukiManager::processStreamData, *this) // establish the callback
  {
    kobuki::Parameters parameters;
    parameters.sigslots_namespace = "/kobuki"; // configure the first part of the sigslot
namespace
    parameters.device_port = "/dev/kobuki";         // the serial port to connect to (windows
COM1..)
    kobuki.init(parameters);
    slot_stream_data.connect("/kobuki/stream_data");
  }

  void spin() {
    ecl::Sleep sleep(1);
    while ( true ) {
      sleep();
    }
  }

  /*
 Called whenever the kobuki receives a data packet. Up to you from here to process it.

 Note that special processing is done for the various events which discretely change
 state (bumpers, cliffs etc) and updates for these are informed via the xxxEvent
 signals provided by the kobuki driver.
   */
  void processStreamData() {
    kobuki::CoreSensors::Data data = kobuki.getCoreSensorData();
    std::cout << "Encoders [" <<  data.left_encoder << "," << data.right_encoder << "]" <<
std::endl;
  }

private:
  kobuki::Kobuki kobuki;
  ecl::Slot<> slot_stream_data;
};

int main() {
  KobukiManager kobuki_manager;
  kobuki_manager.spin();
  return 0;
}
```

Other Signals

The other signals will pass a structure of a particular type with the transmitted information. Process the same way, using the event type as the argument to the callback function.

Detailed Example

More detailed example code can be found in the ros kobuki node implementation. See the kobuki_node implementation for a ros platform.

Troubleshooting

While debugging, you may often accidentally leave sigslots dangling, typos for the connection name are a common cause. For this, there is an introspection method available which you can use to quickly print the currently sigslot connections (dangling or otherwise).

A code snippet:

```
  Kobuki kobuki
  Parameters parameters;
  // configure parameters here
  kobuki.init(parameters);
  // make some sigslot connections here
  kobuki.printSigSlotConnections();
```

Depending on your sigslot connection configuration, you should see something like the following,

```
========== Void ==========
Topics
  Name: /kobuki/stream_data
    # Subscribers: 1
    # Publishers : 1
========= String =========
Topics
  Name: /kobuki/ros_debug
    # Subscribers: 1
    # Publishers : 1
  Name: /kobuki/ros_error
    # Subscribers: 1
    # Publishers : 2
  Name: /kobuki/ros_info
    # Subscribers: 1
    # Publishers : 1
  Name: /kobuki/ros_warn
    # Subscribers: 1
    # Publishers : 2
====== Button Event ======
Topics
  Name: /kobuki/button_event
    # Subscribers: 1
    # Publishers : 1
====== Bumper Event ======
Topics
  Name: /kobuki/bumper_event
    # Subscribers: 1
    # Publishers : 1
...
```

This uses the sigslots manager to retrieve the information. A full example of its use can be found in the ecl_sigslot sources: example cpp program.