1 Wiki

1.1 Einleitung

Das folgende Kapitel ist so geschrieben, dass es möglichst einfach in das EEROS-Wiki[1] übernommen werden kann. Da das EEROS-Wiki in englischer Sprache geschrieben ist, sind auch die folgenden Kapitel in Englisch verfasst.

1.2 Hello World!

1.2.1 ROS with HAL or as block in the control System

1.2.2 Configure the toolchain

To build an EEROS application with ROS, ROS ”kinetic’ needs to be installed[2] on the developer machine and the target machine. Before a ROS application can be started, you need to run the *setup.sh* script of ROS. The same applies for building the EEROS-library with ROS-support and for building an EEROS-application with ROS-support.

In EEROS *CMAKE* is used to build an application. If the EEROS application has dependencies on ROS, the *setup.sh* script of ROS has to be executed before *CMAKE* is called. If an IDE like ”*Qt Creator*” is used, the software has to be started from a terminal. *CMAKE* will not find the ROS library, if *QT Creator* is launched from a desktop icon.

[1]http://wiki.eeros.org/
[2]TODO

1.2.3 Configure the CMAKE file

The following example shows a *CMAKE* file for a simple EEROS application with ROS.

```
cmake_minimum_required(VERSION 2.8)

project(helloWorld)


## ROS
## ///////////////////////////////////////////////////////////////////////
message(STATUS "looking for package 'ROS'")
find_package( roslib REQUIRED )
if (roslib_FOUND)
  message( STATUS "-> ROS found")
  include_directories( "${roslib_INCLUDE_DIRS}" )
  message( STATUS "roslib_INCLUDE_DIRS: " ${roslib_INCLUDE_DIRS} )
  list(APPEND ROS_LIBRARIES "${roslib_LIBRARIES}")
  find_package( rosconsole REQUIRED)
  list(APPEND ROS_LIBRARIES "${rosconsole_LIBRARIES}")
  find_package( roscpp REQUIRED )
  list(APPEND ROS_LIBRARIES "${roscpp_LIBRARIES}")
else()
  message( STATUS "-> ROS NOT found")
endif()


## EEROS
## ///////////////////////////////////////////////////////////////////////
find_package(EEROS REQUIRED)
include_directories(${EEROS_INCLUDE_DIR})
link_directories(${EEROS_LIB_DIR})


## Application
## ///////////////////////////////////////////////////////////////////////
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")

```

```
add_executable(helloWorld
  main.cpp
)


target_link_libraries(helloWorld eeros ${ROS_LIBRARIES})
```

1.3 EEROS with blocks as interface

1.3.1 Introduction

1.3.2 Using a ROS block in the Control System

1.3.3 Creating a new ROS subscriber block

Creating a new Block

A-1 Include the header file of the ROS message.

A-2 Define the type of the ROS message.

A-3 Name your block and create the constructor.

A-4 Add a new EEROS output for everey datafield you want to read -> see below.

Adding a new data field an an EEROS output

B-1 Create EEROS outputs.

B-2 Add a 'getOutput()' function for each output.

B-3 Set the timestamp of all EEROS signal.

B-4 For data fields of variable length use EEROS matrices -> see below.

Adding a new data field of variable length and an EEROS matrix output

C-1 Create the template definition. Each EEROS matrix output needs its own type.

C-2 Create a 'value' and an 'output' variable for each EEROS matrix output.

C-3 Add a 'getOutput()' function for each EEROS matrix output.

C-4 Convert the vector of the ROS message to a <double>-vector.

C-5 Fill the vector into an EEROS matrix.

C-6 Fill the EEROS matrix in a EEROS signal.

1.3.4 Creating a new ROS publisher block

Creating a new Block

Identical precure to 1.3.3 "Creating a new Block";

Adding a new data field an an EEROS input

B-1 Create EEROS inputs.

B-2 Add a 'getInput()' function for each input.

B-3 If available, set time in msg header.

B-4 Check if EEROS input is connected. Cast the data. Assign casted data to ROS message field.

B-5 For data fields of variable length use EEROS matrices -> see below.

Adding a new data field of variable length and an EEROS matrix input

C-1 Create the template definition. Each EEROS matrix input needs its own type.

C-2 Create a 'value' and an 'output' variable for each EEROS matrix output.

C-3 Add a 'getInput()' function for each EEROS matrix Input.

C-4 Check if EEROS input is connected.

C-5 Get the vector from the EEROS input.

C-6 Cast the vector and assign it to the appropriate ROS data field.

1.4 EEROS HAL with ROS

1.4.1 Introduction

The wrapper library ”*ros-eeros*” is used to connect the EEROS HAL with ROS topics. With the EEROS HAL digital and analougue inputs and outputs can be defined with an *.json file. If you want to test your application with an *Gazebo* simulation, you can define your inputs and outputs as ROS-topics to connect your application with the simmulation completly without any real hardware. To use your application with hardware, you can use for example the wrapper library *comedi-eeros* ord *flink-eeros*. If you adapt the *.json file correctly, your application should now run on hardware with real encoders and motors without any problems.

It is also possible, to use ROS-topics alongside real hardware. You can use *comedi-eeros* to read an encoder and set a control value for a motor. At the same time you can publish the same values on rostopics to monitor them with *Gazebo* (if you have a model of your robot) or you can monitor the values with a ROS-tool like ”*rqt*” in graph with *MatPlot* or as numers in the *TopicMonitor*. To store these values you can use *ROS-Bag*.

There are hundreds of different message types in ROS and it is possible to create custom types. Because every message type has be handled differently, only a few are suported by default. But the wrapper library can easily be extendendet to support addititional message types.

Tabelle 1.1: Most important key-value pairs for ros-eeros

Key	Typical value	Description
library	libroseeros.so	Wrapper library for ROS
devHandle	testNodeHAL	ROS node created by HAL
type	AnalogIn / AnalogOut / DigIn / DigOut	Type of input
additionalArguments	'see next table'	'see next table'

Tabelle 1.2: Additional arguments specific for ros-eeros

Key	Typical value	Description
topic	/testNode/TestTopic1	Topic to listen / subscribe
msgType	sensor_msgs::LaserScan	ROS message type of topic
dataField	scan_time	Desired data member of message
callOne	true	Oldest, not yet fetched message is fetched
callOne	false	Newest available message is fetched
queueSize	1000	Size of buffer; queueSize=1000 if omitted

1.4.2 The *.json file

Table 1.1 shows the most important key-values pairs for using the HAL with ROS.

The *additionalArguments* are special arguments which are parsed in the wrapper library *ros-eeros*. These arguments contain additional informations, which are necessary to communicate with a ROS network. All arguments are separated with a semicolon. The available arguments are listed in table 1.2.

An example for the additional arguments could be:

```
"additionalArguments": "topic=/testNode/TestTopic3; msgType=sensor_msgs::LaserScan;
dataField=scan_time; callOne=false; queueSize=100",
```

Table 1.2 shows all currently available *additionalArguments*. **topic** and **msgType** are mandatory arguments.

In table 1.3 are all currently implemented message types and associated data fields. If your desired message type is not implemented yet, you can easyly implement it yourself. See chapter 1.6 for an guide to implement additional message types and data fields in *ros-eeros*.

You can find a complete example, including a *.json file, in the eeros framework (/examples/hal/Ros*).

Tabelle 1.3: Currently implemented message types in ros-eeros

HAL type	msgType	dataField
AnalogIn	std_msgs::Float64	-
	sensor_msgs::LaserScan	angle_min
		angle_max
		angle_increment
		time_increment
		scan_time
		range_min
		range_max
AnalogOut	std_msgs::Float64	-
	sensor_msgs::LaserScan	angle_min
		angle_max
		angle_increment
		time_increment
		scan_time
		range_min
		range_max
DigIn	sensor_msgs::BatteryState	present
DigOut	sensor_msgs::BatteryState	present

1.5 Using the ROS HAL in an EEROS application

Refere to the documentation of the EEROS HAL[3] and check the example in the eeros framework (/examples/hal/Ros*).

1.6 Add new ROS message type to the HAL

1.6.1 Introduction

First you need to checkout the master branch of the wrapper library[4]. After you have implemented and testet your additions, don't hesitate to push your changes to master.

1.6.2 Add an input

In this example I will describe only an *AnalogIn*. The procedure to create a new msg type and data field for a *DigIn* is similar.

AnalogIn.hpp:

[3] http://wiki.eeros.org/eeros_architecture/hal/start?s[]=hal
[4] https://github.com/eeros-project/sim-eeros

1. Include ROS message type

```
#include <sensor_msgs/LaserScan.h>
```

2. Create new callback functions for ROS

```
void sensorMsgsLaserScanAngleMin (const sensor_msgs::LaserScan::Type& msg)

data = msg.angle_min; ;
```

AnalogIn.cpp:

3. Extend parser by selecting correct callback function for ros

```
else if ( msgType == "sensor_msgs::LaserScan") {

        if ( dataField == "angle_min") subscriber = rosNodeHandle->subscribe(topic, queue-

Size, &AnalogIn::sensorMsgsLaserScanAngleMin, this);
```

1.6.3 Add an output

In this example I will describe only an *AnalogOut*. The procedure to create a new msg type and data field for a *DigOut* is similar.

AnalogOut.hpp:

1. Include ROS message type

```
#include <sensor_msgs/LaserScan.h>
```

2. Declare set function for ROS

```
static void sensorMsgsLaserScanAngleMin (const double value, const ros::Publisher& publis-

her);
```

3. Extend parser by setting callback function

```
else if ( msgType == "sensor_msgs::LaserScan") {
        publisher = rosNodeHandle->advertise<sensor_msgs::LaserScan>(topic, queueSize);
        if ( dataField == "angle_min")
                setFunction = &sensorMsgsLaserScanAngleMin;
```

4. Create set function for ROS

```
void AnalogOut::sensorMsgsLaserScanAngleMin(const double value, const ros::Publisher& pu-
blisher)
{
        sensor_msgs::LaserScan msg;
        msg.header.stamp = getTime();
        msg.angle_min = value;
        publisher.publish(msg);
}
```