# regaccess - Simple PC to uC communication

*mru*

**Abstract**

`regaccess` provides a simple mechanism for remote method invocation from a PC to a microcontroller by generating code for the controller and the host application (currently for python and c#). Besides the function calling capabilities it also provides a multi-type register based programming model, allowing easy communication with automatic persistance of register values.

## Contents

# 1 Introduction

`regaccess` implements two ways to establish communication between a PC (master) and a micro controller (slave). Firstly, a register abstraction and secondly a function invocation abstraction is provided. Communication is done over a serial (RS232) line.

Available registers and callable functions are defined in an simple XML format and code for both sides (slaver and master) is generated.

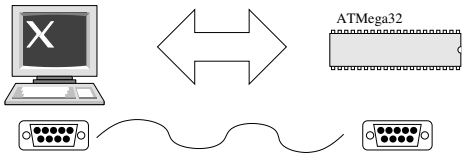


Fig. 1: Schematic

# 2 Architectural Overview

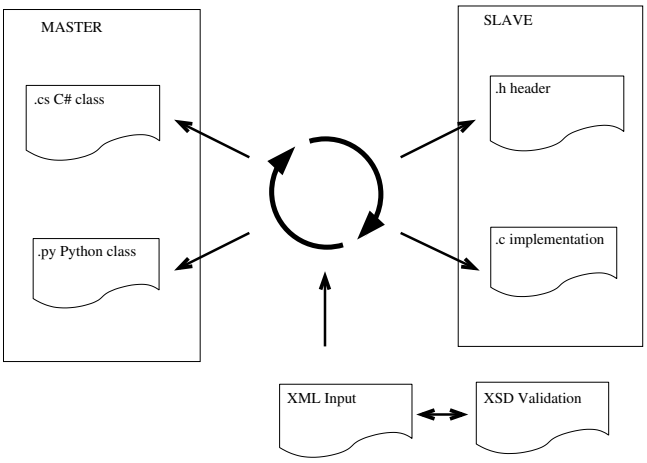


Fig. 2: Architecture

The program is implemented in Python 2.5. Mako templates[1] are used to generate the output. Table 1 on the following page shows the used software component versions.

# 3 Register Model

Several *Register Files* can be defined. Each File defines the type of the contained registers. Access modifiers can be defined for individual registers. Table 3 shows the supported properties, the details are described in Section 5.

[1] `www.makotemplates.org/`

| Component | Version |
|---|---|
| `avr-gcc` | 4.3.4 |
| `python` | 2.6.5 |
| `mako` | 0.2.5 |

Tab. 1: Version overview

| Property | Semantic |
|---|---|
| `type` | register type |
| `init` | initial value for registers |

Tab. 2: Register File Properties

## 3.1 EEPROM Persistence

# 4 Function Invokation

# 5 Input Specification

The XML input is specified with a XML Schema Defintion file. See B on page 6.

# 6 Code Generation

Mako is used to generate the output code. See C on page 8.

# 7 Serial Communication Protocol

A Master-Slave communication protocol is implemented. All communication is initiated by the master (the PC). The slave (microcontroller) listens to the serial port and reacts to the requests by the master.

Datagrams are sent byte-wise and for efficiency no strings are used.

A datagram sent by the master is sent as follows:

- An opcode is sent. This is one byte long
- in-parameters of the functions are transmitted.
- The slave responses with a status code
- IF the status code is `STATUS_OK` AND the called function has out-parameters
  - the out-parameters are transmitted from the slave to the host

| Property | Semantic | Default Value |
|---|---|---|
| `read` | allow read access | 1 |
| `write` | allow write access | 1 |
| `persist` | automatic EEPROM persitence | 0 |
| `default` | assign default value | n/a |

Tab. 3: Register Properties

## 7.1 Marshalling

Supported Types:

- float
- byte
- short
- ushort

## 7.2 Status Codes

# A Input Example

Listing 1: Input Example

```
<?xml version="1.0" encoding="utf-8" ?>
<regaccess
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="regaccess.xsd"
           if_version="2">

  <file type="byte" init="0">
  </file>

  <file type="float" init="0.0">
    <register id="0" name="kp" default="1.0" persist="1">
      <doc>propotional factor for pid controller</doc>
    </register>
    <register id="1" name="ki" default="0.0" persist="1">
      <doc>integral factor for pid controller</doc>
    </register>
    <register id="2" name="kd" default="0.0" persist="1"/>
    <register id="3" name="abgas_v1190" default="69.553e-3"/>
    <register id="4" name="abgas_amp_gain" default="50.0"/>
    <register id="5" name="vorlauf_v1190" default="69.553e-3"/>
    <register id="6" name="vorlauf_amp_gain" default="50.0"/>
  </file>
```

```
  <file type="short" init="0">
    <register id="0" name="temp_vorlauf" default="0"
        write="0"/>
    <register id="1" name="temp_abgas" default="0" write="0" />
    <register id="2" name="temp_ambient" default="25"
        write="0" persist="1"/>
    <register id="3" name="controller_output" default="0"
        write="0" />
  </file>

  <file type="ushort" init="0">
  </file>

  <status>
    <statuscode id="0" name="OK"/>
    <statuscode id="1" name="FAIL"/>
    <statuscode id="2" name="NO_SUCH_REGISTER"/>
    <statuscode id="3" name="NO_ACCESS"/>
    <statuscode id="4" name="INVALID_OPCODE"/>
    <statuscode id="5" name="NOT_IMPLEMENTED"/>
    <statuscode id="6" name="PONG"/>
  </status>

  <functions>

    <function name="write_float_register">
      <param type="byte" name="id"/>
      <param type="float" name="value"/>
    </function>

    <function name="write_short_register">
      <param type="byte" name="id"/>
      <param type="short" name="value"/>
    </function>
<!--
    <function name="write_byte_register">
      <param type="byte" name="id"/>
      <param type="byte" name="value"/>
    </function>
-->
    <function name="read_float_register">
      <param type="byte" name="id"/>
      <param type="float" name="value" direction="out"/>
    </function>

    <function name="read_short_register">
      <param type="byte" name="id"/>
      <param type="short" name="value" direction="out"/>
    </function>
<!--
    <function name="read_byte_register">
      <param type="byte" name="id"/>
      <param type="byte" name="value" direction="out"/>
    </function>
```

```
-->

    <function name="set_led">
      <param type="byte" name="on"/>
    </function>

    <function name="get_if_version">
      <param type="byte" name="version" direction="out"/>
    </function>

    <function name="ping"/>

    <function name="ln5623_set_output">
      <param type="ushort" name="value"/>
      <param type="byte" name="dp"/>
    </function>
  </functions>
</regaccess>
```

# B Input Validation

Listing 2: The schema definition for the XML files

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:annotation>
    <xs:documentation xml:lang="en">
      Schema for regaccess xml files
    </xs:documentation>
  </xs:annotation>

  <xs:element name="regaccess" type="RegisterType"/>

  <xs:complexType name="FileType">
    <xs:sequence minOccurs="0" maxOccurs="100">
      <xs:element name="register">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="doc" type="xs:string"
                minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="id" type="xs:unsignedInt"
              use="required"/>
          <xs:attribute name="name" type="xs:Name"
              use="required"/>
          <xs:attribute name="default" type="xs:string"/>
          <xs:attribute name="persist" type="xs:boolean"
              default="0"/>
          <xs:attribute name="write" type="xs:boolean"
              default="1"/>
          <xs:attribute name="read" type="xs:boolean"
              default="1"/>
        </xs:complexType>
```

```
    </xs:element>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string"/>

  <!-- default value for initializing local variables -->
  <xs:attribute name="init" type="xs:string"/>
</xs:complexType>

<xs:complexType name="StatusType">
  <xs:sequence>
    <xs:element name="statuscode" minOccurs="0"
        maxOccurs="100">
      <xs:complexType>
        <xs:attribute name="id" type="xs:unsignedInt"
            use="required"/>
        <xs:attribute name="name" type="xs:Name"
            use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="FunctionsType">
  <xs:sequence>
    <xs:element name="function" minOccurs="0"
        maxOccurs="100">
      <xs:complexType>

        <xs:sequence>
          <xs:element name="param" minOccurs="0"
              maxOccurs="100">
            <xs:complexType>
              <xs:attribute name="name" type="xs:Name"
                  use="required"/>
              <xs:attribute name="type" type="xs:Name"
                  use="required"/>
              <xs:attribute name="direction" default="in">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="in"/>
                    <xs:enumeration value="out"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
        </xs:sequence>

        <xs:attribute name="name" type="xs:Name"
            use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

```
  <xs:complexType name="RegisterType">
    <xs:sequence>
      <xs:element name="file" type="FileType" minOccurs="0"
          maxOccurs="100"/>
      <xs:element name="status" type="StatusType"/>
      <xs:element name="functions" type="FunctionsType"/>
    </xs:sequence>
    <xs:attribute name="if_version" type="xs:unsignedInt"
        use="required"/>
  </xs:complexType>

</xs:schema>

<!--  LocalWords:  regaccess
  -->
```

# C A Mako Template

Listing 3: Template to generate the slave implementation

```
<%!
def gen_actual_params(fu):
    return ",".join( map(lambda p: ('&' if p.direction=='out'
        else '') + p.name, fu))

def gen_formal_params(fu):
    if len(fu) == 0:
        return "void"
    return ",".join( map(lambda p: p.type + "␣" + ('*' if
        p.direction=='out' else '') + p.name, fu))
%><%
init_values = {}
for fi in files:
  init_values[fi.type] = fi.init
%>\
#include <avr/eeprom.h>

#include "common.h"

#define EEPROM_VALID_MASK 0xA5

enum {
% for o in opcodes:
REG_OP_${o},
% endfor
};

% for k,v in statuscodes.items():
#define REG_ST_${v}  ${k}
% endfor

byte ping(void) { return 0; }
```

```
byte get_fw_version(void) { return 0; }
byte get_if_version(byte* version) { return ${if_version}; }


// ----------------------- file definitions

// register in memory layout
typedef struct {
% for fi in files:
% for r in fi.entries:
  ${fi.type} ${r.name};     // ${r.doc}
% endfor
% endfor
} reg_file_t;

volatile reg_file_t reg_file;


// persistance of variables: eeprom layout

typedef struct {
% for fi in files:
% for r in filter(lambda entry: entry.persist, fi.entries):
  ${fi.type} ${r.name};
% endfor
% endfor
} reg_file_persist_t;


reg_file_persist_t reg_persist_eeprom EEMEM;
uint8_t ee_valid_configuration EEMEM;


static void __attribute__((constructor))
regfile_autoinit(void)
{
% for fi in files:
% for r in filter(lambda entry: entry.default, fi.entries):
  reg_file.${r.name} = ${r.default};
% endfor
% endfor

  if ( eeprom_read_byte(&ee_valid_configuration) ==
      EEPROM_VALID_MASK) {

% for fi in files:
% for r in filter(lambda entry: entry.persist, fi.entries):
    eeprom_read_block ((void *) &reg_file.${r.name},
                       (const void *)
                           &reg_persist_eeprom.${r.name},
                       sizeof(${fi.type}));
% endfor
% endfor
  }
```

```
  else {
% for fi in files:
% for r in filter(lambda entry: entry.persist, fi.entries):
    eeprom_write_block( (const void*) &reg_file.${r.name},
                        &reg_persist_eeprom.${r.name},
                        sizeof(${fi.type}));
% endfor
% endfor
    eeprom_write_byte(&ee_valid_configuration,
        EEPROM_VALID_MASK);
  }
}



// ----------------------- file accessors

% for fi in files:
% for r in fi.entries:

extern ${fi.type} get_${r.name}(void) { return
    reg_file.${r.name}; }
extern void  set_${r.name}(const ${fi.type} v) {
  reg_file.${r.name} = v;
% if r.persist:
  eeprom_write_block( (const void*) &reg_file.${r.name},
                      &reg_persist_eeprom.${r.name},
                      sizeof(${fi.type}));
% endif
}
% endfor
% endfor

% for fi in filter(lambda f: len(f.entries) > 0, files):
static byte read_${fi.type}_register(const byte id,
    ${fi.type}* value)  {
  switch(id) {
% for r in filter(lambda entry: entry.read, fi.entries):
  case ${r.id}: *value = get_${r.name}(); return REG_ST_OK;
     break;
% endfor
% for r in filter(lambda entry: not entry.read, fi.entries):
  case ${r.id}: return REG_ST_NO_ACCESS; break;
% endfor
  }
  return REG_ST_NO_SUCH_REGISTER;
}

static byte write_${fi.type}_register(const byte id, const
    ${fi.type} value)  {
  switch(id) {
% for r in filter(lambda entry: entry.write, fi.entries):
  case ${r.id}: set_${r.name}(value); return REG_ST_OK; break;
% endfor
```

```
% for r in filter(lambda entry: not entry.write, fi.entries):
  case ${r.id}: return REG_ST_NO_ACCESS; break;
% endfor
  }
  return REG_ST_NO_SUCH_REGISTER;
}

% endfor


// ---------------------- serial comm accessors


static byte receive_byte(void) {
  unsigned int v;
  while ( (v=uart_getc()) & UART_NO_DATA );
  return (byte) v&0xff;
}

static float receive_float(void) {
  union { char b[4]; float f; } v;
  v.b[0] = receive_byte();
  v.b[1] = receive_byte();
  v.b[2] = receive_byte();
  v.b[3] = receive_byte();
  return v.f;
}

static short receive_short(void) {
  union { char b[2]; short f; } v;
  v.b[0] = receive_byte();
  v.b[1] = receive_byte();
  return v.f;
}

static short receive_ushort(void) {
  union { char b[2]; ushort f; } v;
  v.b[0] = receive_byte();
  v.b[1] = receive_byte();
  return v.f;
}

static void send_byte(const byte value) {
  uart_putc(value);
}

static void send_float(const float value) {
  union { char b[4]; float f; } v;
  v.f = value;

  send_byte( v.b[0] );
  send_byte( v.b[1] );
  send_byte( v.b[2] );
```

```
  send_byte( v.b[3] );
}

static void send_short(const short value) {
  union { char b[2]; short f; } v;
  v.f = value;

  send_byte( v.b[0] );
  send_byte( v.b[1] );
}

static void send_ushort(const short value) {
  union { char b[2]; ushort f; } v;
  v.f = value;

  send_byte( v.b[0] );
  send_byte( v.b[1] );
}

extern void on_receive_byte(const uint8_t ch) {

  byte status = 0;

  switch(ch) {
% for fu in fus:
  case REG_OP_${fu.name}: {
% for p in filter(lambda p: p.direction=="in", fu.params):
      ${p.type} ${p.name} = receive_${p.type}();
% endfor
% for p in filter(lambda p: p.direction=="out", fu.params):
      ${p.type} ${p.name} = ${init_values[p.type]};
% endfor

      status = ${fu.name}(${gen_actual_params(fu.params)});

      send_byte(status);
% if len(filter(lambda p: p.direction=="out", fu.params)) > 0:
      if ( REG_ST_OK == status ) {
% for p in filter(lambda p: p.direction=="out", fu.params):
        send_${p.type}(${p.name});
% endfor
      }
% endif
    }
    break;
% endfor

  default:
    send_byte(REG_ST_INVALID_OPCODE);
  }
}
```

# D Generated Code examples: Slave header

Listing 4: Slave header

```
#ifndef _regaccess_h
#define _regaccess_h

typedef uint8_t byte;
typedef uint16_t ushort;

// ---------------------- accessors


extern float get_kp(void);
extern void  set_kp(const float f);

extern float get_ki(void);
extern void  set_ki(const float f);

extern float get_kd(void);
extern void  set_kd(const float f);

extern float get_abgas_v1190(void);
extern void  set_abgas_v1190(const float f);

extern float get_abgas_amp_gain(void);
extern void  set_abgas_amp_gain(const float f);

extern float get_vorlauf_v1190(void);
extern void  set_vorlauf_v1190(const float f);

extern float get_vorlauf_amp_gain(void);
extern void  set_vorlauf_amp_gain(const float f);

extern short get_temp_vorlauf(void);
extern void  set_temp_vorlauf(const short f);

extern short get_temp_abgas(void);
extern void  set_temp_abgas(const short f);

extern short get_temp_ambient(void);
extern void  set_temp_ambient(const short f);

extern short get_controller_output(void);
extern void  set_controller_output(const short f);

// ---------------------- serial comm accessors

extern void on_receive_byte(const uint8_t ch);



// ---------------------- declarations of external functions

extern byte set_led(byte on);
extern byte ping(void);
extern byte ln5623_set_output(ushort value,byte dp);
```

```
#endif
```

# E Generated Code examples: Slave implementation

Listing 5: Slave implementation

```
#include <avr/eeprom.h>

#include "common.h"

#define EEPROM_VALID_MASK 0xA5

enum {
REG_OP_write_float_register,
REG_OP_write_short_register,
REG_OP_read_float_register,
REG_OP_read_short_register,
REG_OP_set_led,
REG_OP_get_if_version,
REG_OP_ping,
REG_OP_ln5623_set_output,
};

#define REG_ST_FAIL  1
#define REG_ST_OK  0
#define REG_ST_NO_ACCESS  3
#define REG_ST_NO_SUCH_REGISTER  2
#define REG_ST_NOT_IMPLEMENTED  5
#define REG_ST_INVALID_OPCODE  4
#define REG_ST_PONG  6

byte ping(void) { return 0; }

byte get_fw_version(void) { return 0; }
byte get_if_version(byte* version) { return 2; }


// ---------------------- file definitions

// register in memory layout
typedef struct {
  float kp;     //
  float ki;     //
  float kd;     //
  float abgas_v1190;     //
  float abgas_amp_gain;     //
  float vorlauf_v1190;     //
  float vorlauf_amp_gain;     //
  short temp_vorlauf;     //
  short temp_abgas;     //
  short temp_ambient;     //
  short controller_output;     //
} reg_file_t;
```

```
volatile reg_file_t reg_file;


// persistance of variables: eeprom layout

typedef struct {
  float kp;
  float ki;
  float kd;
  short temp_ambient;
} reg_file_persist_t;


reg_file_persist_t reg_persist_eeprom EEMEM;
uint8_t ee_valid_configuration EEMEM;


static void __attribute__((constructor))
regfile_autoinit(void)
{
  reg_file.kp = 1.0;
  reg_file.ki = 0.0;
  reg_file.kd = 0.0;
  reg_file.abgas_v1190 = 69.553e-3;
  reg_file.abgas_amp_gain = 50.0;
  reg_file.vorlauf_v1190 = 69.553e-3;
  reg_file.vorlauf_amp_gain = 50.0;
  reg_file.temp_vorlauf = 0;
  reg_file.temp_abgas = 0;
  reg_file.temp_ambient = 25;
  reg_file.controller_output = 0;

  if ( eeprom_read_byte(&ee_valid_configuration) ==
      EEPROM_VALID_MASK) {

    eeprom_read_block ((void *) &reg_file.kp,
                       (const void *) &reg_persist_eeprom.kp,
                       sizeof(float));
    eeprom_read_block ((void *) &reg_file.ki,
                       (const void *) &reg_persist_eeprom.ki,
                       sizeof(float));
    eeprom_read_block ((void *) &reg_file.kd,
                       (const void *) &reg_persist_eeprom.kd,
                       sizeof(float));
    eeprom_read_block ((void *) &reg_file.temp_ambient,
                       (const void *)
                           &reg_persist_eeprom.temp_ambient,
                       sizeof(short));
  }
  else {
    eeprom_write_block( (const void*) &reg_file.kp,
                        &reg_persist_eeprom.kp,
                        sizeof(float));
```

```
    eeprom_write_block( (const void*) &reg_file.ki,
                        &reg_persist_eeprom.ki,
                        sizeof(float));
    eeprom_write_block( (const void*) &reg_file.kd,
                        &reg_persist_eeprom.kd,
                        sizeof(float));
    eeprom_write_block( (const void*) &reg_file.temp_ambient,
                        &reg_persist_eeprom.temp_ambient,
                        sizeof(short));
    eeprom_write_byte(&ee_valid_configuration,
        EEPROM_VALID_MASK);
  }
}



// ----------------------- file accessors


extern float get_kp(void) { return reg_file.kp; }
extern void  set_kp(const float v) {
  reg_file.kp = v;
  eeprom_write_block( (const void*) &reg_file.kp,
                      &reg_persist_eeprom.kp,
                      sizeof(float));
}

extern float get_ki(void) { return reg_file.ki; }
extern void  set_ki(const float v) {
  reg_file.ki = v;
  eeprom_write_block( (const void*) &reg_file.ki,
                      &reg_persist_eeprom.ki,
                      sizeof(float));
}

extern float get_kd(void) { return reg_file.kd; }
extern void  set_kd(const float v) {
  reg_file.kd = v;
  eeprom_write_block( (const void*) &reg_file.kd,
                      &reg_persist_eeprom.kd,
                      sizeof(float));
}

extern float get_abgas_v1190(void) { return
    reg_file.abgas_v1190; }
extern void  set_abgas_v1190(const float v) {
  reg_file.abgas_v1190 = v;
}

extern float get_abgas_amp_gain(void) { return
    reg_file.abgas_amp_gain; }
extern void  set_abgas_amp_gain(const float v) {
  reg_file.abgas_amp_gain = v;
}
```

```
extern float get_vorlauf_v1190(void) { return
    reg_file.vorlauf_v1190; }
extern void  set_vorlauf_v1190(const float v) {
  reg_file.vorlauf_v1190 = v;
}

extern float get_vorlauf_amp_gain(void) { return
    reg_file.vorlauf_amp_gain; }
extern void  set_vorlauf_amp_gain(const float v) {
  reg_file.vorlauf_amp_gain = v;
}

extern short get_temp_vorlauf(void) { return
    reg_file.temp_vorlauf; }
extern void  set_temp_vorlauf(const short v) {
  reg_file.temp_vorlauf = v;
}

extern short get_temp_abgas(void) { return
    reg_file.temp_abgas; }
extern void  set_temp_abgas(const short v) {
  reg_file.temp_abgas = v;
}

extern short get_temp_ambient(void) { return
    reg_file.temp_ambient; }
extern void  set_temp_ambient(const short v) {
  reg_file.temp_ambient = v;
  eeprom_write_block( (const void*) &reg_file.temp_ambient,
                      &reg_persist_eeprom.temp_ambient,
                      sizeof(short));
}

extern short get_controller_output(void) { return
    reg_file.controller_output; }
extern void  set_controller_output(const short v) {
  reg_file.controller_output = v;
}

static byte read_float_register(const byte id, float* value)  {
  switch(id) {
  case 0: *value = get_kp(); return REG_ST_OK; break;
  case 1: *value = get_ki(); return REG_ST_OK; break;
  case 2: *value = get_kd(); return REG_ST_OK; break;
  case 3: *value = get_abgas_v1190(); return REG_ST_OK; break;
  case 4: *value = get_abgas_amp_gain(); return REG_ST_OK;
      break;
  case 5: *value = get_vorlauf_v1190(); return REG_ST_OK;
      break;
  case 6: *value = get_vorlauf_amp_gain(); return REG_ST_OK;
      break;
  }
  return REG_ST_NO_SUCH_REGISTER;
```

```
}

static byte write_float_register(const byte id, const float
    value)  {
  switch(id) {
  case 0: set_kp(value); return REG_ST_OK; break;
  case 1: set_ki(value); return REG_ST_OK; break;
  case 2: set_kd(value); return REG_ST_OK; break;
  case 3: set_abgas_v1190(value); return REG_ST_OK; break;
  case 4: set_abgas_amp_gain(value); return REG_ST_OK; break;
  case 5: set_vorlauf_v1190(value); return REG_ST_OK; break;
  case 6: set_vorlauf_amp_gain(value); return REG_ST_OK; break;
  }
  return REG_ST_NO_SUCH_REGISTER;
}

static byte read_short_register(const byte id, short* value)  {
  switch(id) {
  case 0: *value = get_temp_vorlauf(); return REG_ST_OK; break;
  case 1: *value = get_temp_abgas(); return REG_ST_OK; break;
  case 2: *value = get_temp_ambient(); return REG_ST_OK; break;
  case 3: *value = get_controller_output(); return REG_ST_OK;
      break;
  }
  return REG_ST_NO_SUCH_REGISTER;
}

static byte write_short_register(const byte id, const short
    value)  {
  switch(id) {
  case 0: return REG_ST_NO_ACCESS; break;
  case 1: return REG_ST_NO_ACCESS; break;
  case 2: return REG_ST_NO_ACCESS; break;
  case 3: return REG_ST_NO_ACCESS; break;
  }
  return REG_ST_NO_SUCH_REGISTER;
}



// ---------------------- serial comm accessors


static byte receive_byte(void) {
  unsigned int v;
  while ( (v=uart_getc()) & UART_NO_DATA );
  return (byte) v&0xff;
}

static float receive_float(void) {
  union { char b[4]; float f; } v;
  v.b[0] = receive_byte();
  v.b[1] = receive_byte();
```

```
  v.b[2] = receive_byte();
  v.b[3] = receive_byte();
  return v.f;
}

static short receive_short(void) {
  union { char b[2]; short f; } v;
  v.b[0] = receive_byte();
  v.b[1] = receive_byte();
  return v.f;
}

static short receive_ushort(void) {
  union { char b[2]; ushort f; } v;
  v.b[0] = receive_byte();
  v.b[1] = receive_byte();
  return v.f;
}

static void send_byte(const byte value) {
  uart_putc(value);
}

static void send_float(const float value) {
  union { char b[4]; float f; } v;
  v.f = value;

  send_byte( v.b[0] );
  send_byte( v.b[1] );
  send_byte( v.b[2] );
  send_byte( v.b[3] );
}

static void send_short(const short value) {
  union { char b[2]; short f; } v;
  v.f = value;

  send_byte( v.b[0] );
  send_byte( v.b[1] );
}

static void send_ushort(const short value) {
  union { char b[2]; ushort f; } v;
  v.f = value;

  send_byte( v.b[0] );
  send_byte( v.b[1] );
}

extern void on_receive_byte(const uint8_t ch) {

  byte status = 0;

  switch(ch) {
```

```
case REG_OP_write_float_register: {
    byte id = receive_byte();
    float value = receive_float();

    status = write_float_register(id,value);

    send_byte(status);
  }
  break;
case REG_OP_write_short_register: {
    byte id = receive_byte();
    short value = receive_short();

    status = write_short_register(id,value);

    send_byte(status);
  }
  break;
case REG_OP_read_float_register: {
    byte id = receive_byte();
    float value = 0.0;

    status = read_float_register(id,&value);

    send_byte(status);
    if ( REG_ST_OK == status ) {
      send_float(value);
    }
  }
  break;
case REG_OP_read_short_register: {
    byte id = receive_byte();
    short value = 0;

    status = read_short_register(id,&value);

    send_byte(status);
    if ( REG_ST_OK == status ) {
      send_short(value);
    }
  }
  break;
case REG_OP_set_led: {
    byte on = receive_byte();

    status = set_led(on);

    send_byte(status);
  }
  break;
case REG_OP_get_if_version: {
    byte version = 0;

    status = get_if_version(&version);
```

```
      send_byte(status);
      if ( REG_ST_OK == status ) {
        send_byte(version);
      }
    }
    break;
  case REG_OP_ping: {

      status = ping();

      send_byte(status);
    }
    break;
  case REG_OP_ln5623_set_output: {
      ushort value = receive_ushort();
      byte dp = receive_byte();

      status = ln5623_set_output(value,dp);

      send_byte(status);
    }
    break;

  default:
    send_byte(REG_ST_INVALID_OPCODE);
  }
}
```

# F Generated Code examples: Master / CSharp

Listing 6: Master CSharp

```
using System;
using System.IO.Ports;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization;


namespace AVRConnector {

  public class ConnectionException : Exception, ISerializable {
    public ConnectionException(int code) : base(to_msg(code)) {
    }

    private static string to_msg(int code) {
      switch(code) {
      case STATUSCODE_FAIL : return "FAIL [1]";
      case STATUSCODE_OK : return "OK [0]";
      case STATUSCODE_NO_ACCESS : return "NO_ACCESS [3]";
      case STATUSCODE_NO_SUCH_REGISTER : return
          "NO_SUCH_REGISTER [2]";
```

```
      case STATUSCODE_NOT_IMPLEMENTED : return
          "NOT_IMPLEMENTED [5]";
      case STATUSCODE_INVALID_OPCODE : return "INVALID_OPCODE
          [4]";
      case STATUSCODE_PONG : return "PONG [6]";
      default:
        return "unknown code";
      }
    }

    private const int STATUSCODE_FAIL = 1;
    private const int STATUSCODE_OK = 0;
    private const int STATUSCODE_NO_ACCESS = 3;
    private const int STATUSCODE_NO_SUCH_REGISTER = 2;
    private const int STATUSCODE_NOT_IMPLEMENTED = 5;
    private const int STATUSCODE_INVALID_OPCODE = 4;
    private const int STATUSCODE_PONG = 6;

  }

  public class AVRConnection {
    private SerialPort port;

    private const int OPCODE_FN_write_float_register = 0;
    private const int OPCODE_FN_write_short_register = 1;
    private const int OPCODE_FN_read_float_register = 2;
    private const int OPCODE_FN_read_short_register = 3;
    private const int OPCODE_FN_set_led = 4;
    private const int OPCODE_FN_get_if_version = 5;
    private const int OPCODE_FN_ping = 6;
    private const int OPCODE_FN_ln5623_set_output = 7;


    private const int STATUSCODE_FAIL = 1;
    private const int STATUSCODE_OK = 0;
    private const int STATUSCODE_NO_ACCESS = 3;
    private const int STATUSCODE_NO_SUCH_REGISTER = 2;
    private const int STATUSCODE_NOT_IMPLEMENTED = 5;
    private const int STATUSCODE_INVALID_OPCODE = 4;
    private const int STATUSCODE_PONG = 6;

    public AVRConnection() {
      port = new SerialPort();
      port.BaudRate = 38400;
      port.StopBits = StopBits.One;
      port.ReadTimeout = 500;
      port.Open();
    }

      public Dictionary<string, object> read_all() {
        Dictionary<string, object> fields = new
            Dictionary<string, object>();

        fields["kp"] = get_kp();
```

```
        fields["ki"] = get_ki();
        fields["kd"] = get_kd();
        fields["abgas_v1190"] = get_abgas_v1190();
        fields["abgas_amp_gain"] = get_abgas_amp_gain();
        fields["vorlauf_v1190"] = get_vorlauf_v1190();
        fields["vorlauf_amp_gain"] = get_vorlauf_amp_gain();
        fields["temp_vorlauf"] = get_temp_vorlauf();
        fields["temp_abgas"] = get_temp_abgas();
        fields["temp_ambient"] = get_temp_ambient();
        fields["controller_output"] = get_controller_output();
        return fields;
      }
     public float get_kp() {
        float value;
        read_float_register(0, out value);
        return value;
    }
     public void set_kp(float value) {
        write_float_register(0, value);
     }

     public float get_ki() {
        float value;
        read_float_register(1, out value);
        return value;
    }
     public void set_ki(float value) {
        write_float_register(1, value);
     }

     public float get_kd() {
        float value;
        read_float_register(2, out value);
        return value;
    }
     public void set_kd(float value) {
        write_float_register(2, value);
     }

     public float get_abgas_v1190() {
        float value;
        read_float_register(3, out value);
        return value;
    }
     public void set_abgas_v1190(float value) {
        write_float_register(3, value);
     }

     public float get_abgas_amp_gain() {
        float value;
        read_float_register(4, out value);
        return value;
    }
     public void set_abgas_amp_gain(float value) {
```

```
        write_float_register(4, value);
    }

    public float get_vorlauf_v1190() {
        float value;
        read_float_register(5, out value);
        return value;
    }
    public void set_vorlauf_v1190(float value) {
        write_float_register(5, value);
    }

    public float get_vorlauf_amp_gain() {
        float value;
        read_float_register(6, out value);
        return value;
    }
    public void set_vorlauf_amp_gain(float value) {
        write_float_register(6, value);
    }

    public short get_temp_vorlauf() {
        short value;
        read_short_register(0, out value);
        return value;
    }
    public short get_temp_abgas() {
        short value;
        read_short_register(1, out value);
        return value;
    }
    public short get_temp_ambient() {
        short value;
        read_short_register(2, out value);
        return value;
    }
    public short get_controller_output() {
        short value;
        read_short_register(3, out value);
        return value;
    }
```

```
public void WriteByte(byte b) {
  Byte[] x = new Byte[1];
  x[0] = b;
  port.Write(x, 0, 1);
}

public void WriteShort(short b) {
  //
}
```

```
public void WriteUShort(ushort b) {
  //
}

public void WriteFloat(float b) {
  //  Byte[] x = new Byte[4];
  //  x[0] = b;
  //  port.Write(x, 0, 1);
  //tbd
}


public byte ReadByte() {
  Byte[] x = new Byte[1];
  port.Read(x, 0, 1);
  return x[0];
}

public short ReadShort() {
  return 0;
}

public ushort ReadUShort() {
  return 0;
}

public float ReadFloat() {
  return 0.0f;
}



public void write_float_register(byte id, float value) {

  port.DiscardInBuffer();
  port.DiscardOutBuffer();

  WriteByte(OPCODE_FN_write_float_register);
  WriteByte(id);
  WriteFloat(value);

  int status = ReadByte();
  if (status != STATUSCODE_OK)
    throw new ConnectionException(status);
}

public void write_short_register(byte id, short value) {

  port.DiscardInBuffer();
  port.DiscardOutBuffer();

  WriteByte(OPCODE_FN_write_short_register);
  WriteByte(id);
```

```
    WriteShort(value);

    int status = ReadByte();
    if (status != STATUSCODE_OK)
      throw new ConnectionException(status);
  }

  public void read_float_register(byte id, out float value) {

    port.DiscardInBuffer();
    port.DiscardOutBuffer();

    WriteByte(OPCODE_FN_read_float_register);
    WriteByte(id);

    int status = ReadByte();
    if (status != STATUSCODE_OK)
      throw new ConnectionException(status);
    value = ReadFloat();
  }

  public void read_short_register(byte id, out short value) {

    port.DiscardInBuffer();
    port.DiscardOutBuffer();

    WriteByte(OPCODE_FN_read_short_register);
    WriteByte(id);

    int status = ReadByte();
    if (status != STATUSCODE_OK)
      throw new ConnectionException(status);
    value = ReadShort();
  }

  public void set_led(byte on) {

    port.DiscardInBuffer();
    port.DiscardOutBuffer();

    WriteByte(OPCODE_FN_set_led);
    WriteByte(on);

    int status = ReadByte();
    if (status != STATUSCODE_OK)
      throw new ConnectionException(status);
  }

  public void get_if_version(out byte version) {

    port.DiscardInBuffer();
    port.DiscardOutBuffer();

    WriteByte(OPCODE_FN_get_if_version);
```

```
    int status = ReadByte();
    if (status != STATUSCODE_OK)
      throw new ConnectionException(status);
    version = ReadByte();
  }

  public void ping() {

    port.DiscardInBuffer();
    port.DiscardOutBuffer();

    WriteByte(OPCODE_FN_ping);

    int status = ReadByte();
    if (status != STATUSCODE_OK)
      throw new ConnectionException(status);
  }

  public void ln5623_set_output(ushort value, byte dp) {

    port.DiscardInBuffer();
    port.DiscardOutBuffer();

    WriteByte(OPCODE_FN_ln5623_set_output);
    WriteUShort(value);
    WriteByte(dp);

    int status = ReadByte();
    if (status != STATUSCODE_OK)
      throw new ConnectionException(status);
  }


 }
}
```

# G Generated Code examples: Master / Python

Listing 7: Master Python

```
import serial
import struct

__doc__ = """
generated interface file for serial communication with avr
    controller
based on the xml register definitions
"""

OPCODE_FN_write_float_register,OPCODE_FN_write_short_register,OPCODE_FN_read_float_regis
    = range(8)

STATUSCODE_FAIL = 1
```

```
STATUSCODE_OK = 0
STATUSCODE_NO_ACCESS = 3
STATUSCODE_NO_SUCH_REGISTER = 2
STATUSCODE_NOT_IMPLEMENTED = 5
STATUSCODE_INVALID_OPCODE = 4
STATUSCODE_PONG = 6

status_names = {
STATUSCODE_FAIL : 'FAIL [1]',
STATUSCODE_OK : 'OK [0]',
STATUSCODE_NO_ACCESS : 'NO_ACCESS [3]',
STATUSCODE_NO_SUCH_REGISTER : 'NO_SUCH_REGISTER [2]',
STATUSCODE_NOT_IMPLEMENTED : 'NOT_IMPLEMENTED [5]',
STATUSCODE_INVALID_OPCODE : 'INVALID_OPCODE [4]',
STATUSCODE_PONG : 'PONG [6]',
}

class ConnectionException(Exception):
    def __init__(self, code):
        self.code = code

    def __str__(self):
        global status_names
        if self.code in status_names.keys():
            return status_names[self.code]
        return "unknown status code [%d]" % self.code

class RegConnection():
    def __init__(self):
        # configure the serial connections (the parameters
            differs on the device you are connecting to)
        self.ser = serial.Serial(
            port='/dev/ttyS0',
            baudrate=38400,
            parity=serial.PARITY_NONE,
            stopbits=serial.STOPBITS_ONE,
            bytesize=serial.EIGHTBITS,
            timeout=1
            )

        self.ser.open()
        self.ser.isOpen()

    def read_all(self):
        fields = {}
        fields["kp"] = self.get_kp()
        fields["ki"] = self.get_ki()
        fields["kd"] = self.get_kd()
        fields["abgas_v1190"] = self.get_abgas_v1190()
        fields["abgas_amp_gain"] = self.get_abgas_amp_gain()
        fields["vorlauf_v1190"] = self.get_vorlauf_v1190()
        fields["vorlauf_amp_gain"] =
            self.get_vorlauf_amp_gain()
        fields["temp_vorlauf"] = self.get_temp_vorlauf()
```

```
        fields["temp_abgas"] = self.get_temp_abgas()
        fields["temp_ambient"] = self.get_temp_ambient()
        fields["controller_output"] =
            self.get_controller_output()
        return fields

    def get_kp(self):
        return self.read_float_register(0)
    def set_kp(self, value):
        self.write_float_register(0, value)

    def get_ki(self):
        return self.read_float_register(1)
    def set_ki(self, value):
        self.write_float_register(1, value)

    def get_kd(self):
        return self.read_float_register(2)
    def set_kd(self, value):
        self.write_float_register(2, value)

    def get_abgas_v1190(self):
        return self.read_float_register(3)
    def set_abgas_v1190(self, value):
        self.write_float_register(3, value)

    def get_abgas_amp_gain(self):
        return self.read_float_register(4)
    def set_abgas_amp_gain(self, value):
        self.write_float_register(4, value)

    def get_vorlauf_v1190(self):
        return self.read_float_register(5)
    def set_vorlauf_v1190(self, value):
        self.write_float_register(5, value)

    def get_vorlauf_amp_gain(self):
        return self.read_float_register(6)
    def set_vorlauf_amp_gain(self, value):
        self.write_float_register(6, value)

    def get_temp_vorlauf(self):
        return self.read_short_register(0)
    def get_temp_abgas(self):
        return self.read_short_register(1)
    def get_temp_ambient(self):
        return self.read_short_register(2)
    def get_controller_output(self):
        return self.read_short_register(3)


    def write_float_register(self, id, value):

        self.ser.flushInput()
```

```
self.ser.write(struct.pack("=B",
    OPCODE_FN_write_float_register))
self.ser.write(struct.pack("=B", id))
self.ser.write(struct.pack("=f", value))

    ( status, ) = struct.unpack("=B", self.ser.read(1))
    if status != STATUSCODE_OK:
        raise ConnectionException(status)
    return

def write_short_register(self, id, value):

    self.ser.flushInput()
self.ser.write(struct.pack("=B",
    OPCODE_FN_write_short_register))
self.ser.write(struct.pack("=B", id))
self.ser.write(struct.pack("=h", value))

    ( status, ) = struct.unpack("=B", self.ser.read(1))
    if status != STATUSCODE_OK:
        raise ConnectionException(status)
    return

def read_float_register(self, id):

    self.ser.flushInput()
self.ser.write(struct.pack("=B",
    OPCODE_FN_read_float_register))
self.ser.write(struct.pack("=B", id))

    ( status, ) = struct.unpack("=B", self.ser.read(1))
    if status != STATUSCODE_OK:
        raise ConnectionException(status)
    ( value, ) = struct.unpack("=f", self.ser.read(4));
    return value

def read_short_register(self, id):

    self.ser.flushInput()
self.ser.write(struct.pack("=B",
    OPCODE_FN_read_short_register))
self.ser.write(struct.pack("=B", id))

    ( status, ) = struct.unpack("=B", self.ser.read(1))
    if status != STATUSCODE_OK:
        raise ConnectionException(status)
    ( value, ) = struct.unpack("=h", self.ser.read(2));
    return value

def set_led(self, on):

    self.ser.flushInput()
self.ser.write(struct.pack("=B", OPCODE_FN_set_led))
self.ser.write(struct.pack("=B", on))
```

```
        ( status, ) = struct.unpack("=B", self.ser.read(1))
        if status != STATUSCODE_OK:
            raise ConnectionException(status)
        return

    def get_if_version(self, ):

        self.ser.flushInput()
    self.ser.write(struct.pack("=B", OPCODE_FN_get_if_version))

        ( status, ) = struct.unpack("=B", self.ser.read(1))
        if status != STATUSCODE_OK:
            raise ConnectionException(status)
        ( version, ) = struct.unpack("=B", self.ser.read(1));
        return version

    def ping(self, ):

        self.ser.flushInput()
    self.ser.write(struct.pack("=B", OPCODE_FN_ping))

        ( status, ) = struct.unpack("=B", self.ser.read(1))
        if status != STATUSCODE_OK:
            raise ConnectionException(status)
        return

    def ln5623_set_output(self, value, dp):

        self.ser.flushInput()
    self.ser.write(struct.pack("=B",
        OPCODE_FN_ln5623_set_output))
    self.ser.write(struct.pack("=H", value))
    self.ser.write(struct.pack("=B", dp))

        ( status, ) = struct.unpack("=B", self.ser.read(1))
        if status != STATUSCODE_OK:
            raise ConnectionException(status)
        return
```