Ontwerpdocument LPA

Decompressiekamer

|  |  |
| --- | --- |
| Studentnaam | Rafał Grasman |
| Studentnummer | 2217753 (PCN: 289290) |
| SLB-er | Jan Dobbelsteen |

# Documenthistorie

|  |  |  |
| --- | --- | --- |
| **Datum** | **Versie** | **Belangrijke wijziging** |
| 29-06-2015 | 1.0 | Systeem design opgesteld |
| 30-06-2012 | 2.0 | Systeem design verbeterd en afgemaakt |
|  |  |  |
|  |  |  |
|  |  |  |

# Inhoud

[Documenthistorie 2](#_Toc423405628)

[Inhoud 3](#_Toc423405629)

[Inleiding 4](#_Toc423405630)

[PC Systeem 5](#_Toc423405631)

[Klassendiagram 5](#_Toc423405632)

[Message 6](#_Toc423405633)

[DecompressieKamer 6](#_Toc423405634)

[Embedded Systeem 7](#_Toc423405635)

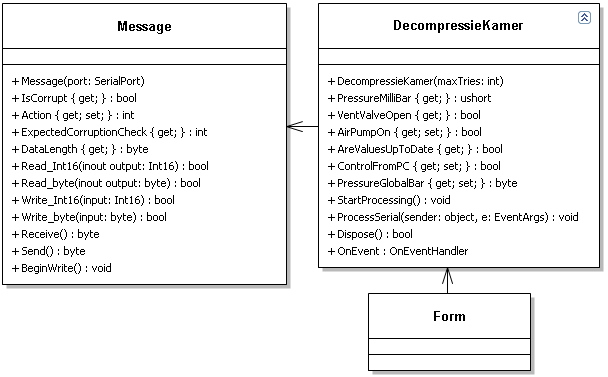
[Protocol 8](#_Toc423405636)

# Inleiding

In dit document worden de klassen en modulen van de C# en C applicaties beschreven voor het LPA.

# PC Systeem

## Klassendiagram



Beschrijving methoden

### Message

Action – de actie die uitgevoerd moet worden.

DataLength – de data lengte die met de message mee is gestuurd.

Expected Corruption Check – de Corruption Check die verwacht wordt.

IsCorrupt – Corruption Check != Excpected Corruption Check.

Send – Stuurt de message via SerialPort, return hoeveelheid bytes verstuurd.

Receive – Ontvangt een message via SerialPoort, return hoeveelheid bytes gelezen.

Write\_T – een variabel (Int16, byte) in de data schrijven en dataLengte aanpassen.

Read\_T – een variabel inlezen uit de data.

BeginWrite – Dit moet worden aangeroepen voordat een nieuwe message voorbereid en gestuurd wordt (actie, data invoegen, etc, sturen)

### DecompressieKamer

PressureGlobalBar – leest de globale druk in bar of set deze.

PressureMilliBar – druk in millibar.

AirPumpOn – return true als de luchtpomp aan staat, anders false.

VentValveOpen – return true als de ontluchtingsklep open staat, anders false.

AreValuesUpToDate – return true als alle vorige waarden minimaal 1 keer geupdate zijn.

ControlFromPC – kies of de PC de besturing doet of het embedded systeem zelf, of check de status

StartProcessing – maakt de timer aan voor ProcessSerial zodat deze in een STA thread kan werken.

ProcessSerial – een System.Windows.Forms.Timer die de seriele data binnenhaalt en wegstuurt zonodig, alle variabelen up to date houd en statusinformatie verwerkt.

Dispose – ruim alles op nadat de klasse niet meer nodig is.

OnEvent – wanneer er een event gebeurd, zoals timeout of nieuwe values update wordt deze uitgevoerd.

# Embedded Systeem

|  |  |
| --- | --- |
| Module naam | Omschrijving |
| Message | Seriele gestructureerde data lezen + schrijven en in bruikbare data omzetten (net als in C# Message klasse) |
| Config | Een eenvoudige globale variabel waarin configuratie bijgehouden kan worden |
| ProcessKeys | Detecteerd key presses en stelt zo de goede druk in mits de configuratie dat toelaat |
| ProcessLCD | Zorgt voor de weergave van de data op de LCD |
| ProcessLeds | Update de leds i.v.m. gemeten druk t.o.v. ingestelde druk |
| ProcessMessages | Verwerkt inkomende data en stuurt data naar de PC |
| ProcessPresurarization | Zorgt voor het correcte drukteniveau |

Message:

void Message\_Init(void);

byte Message\_GetAction(void);

void Message\_SetAction(byte action);

byte Message\_GetDataLenght(void);

bool Message\_Read\_byte(byte\* output);

bool Message\_Write\_byte(byte input);

bool Message\_Read\_Int16(int16\_t\* output);

bool Message\_Write\_Int16(int16\_t input);

byte Message\_Receive(void);//bytes gelezen

byte Message\_Send(void);//bytes verstuurd

bool Message\_IsCorrupt(void);

void Message\_BeginWrite(void);//eerst dit, dan SetAction, dan Write\_\*\_\*, dan Send

Config:

typedef struct

{

bool connected;

byte globalPressureBar;

bool controlledByPC;

} SConfig;

extern SConfig config;

De rest heeft alleen maar Process\*(void) functies.

# Protocol

De communicatie die over seriële verbinding plaats vindt zal voor de meeste betrouwbaarheid een structuur moeten hebben. Daarom wordt het ‘Message’ protocol gedefinieerd:

* 2 ‘Start’ bytes
* Gewenste actie
* De data lengte die mee gestuurd wordt
* 1-byte hash (gelijk aan actie indien data lengte == 0) (Corruption Check)
* Data (indien data lengte > 0)

Om de start bytes te bepalen worden de volgende voorwaarden gesteld (BS – Byte 1 Start, BE – Byte 2 Start):

* BS != BE
* BS != ~BE
* BS > 127 && BE > 127
* NumberOfEnabledBits(max(BS,BE)-min(BS,BE)) > 3

De getallen 230 (0xE6) en 201 (0xC9) zijn hiervoor geschikt omdat deze aan alle 4 voorwaarden voldoen (verschil in bits is 5).

De 1-byte hash wordt berekent door alle data variabelen bij elkaar op te tellen in een int als in:

int sum = 0; for(int i = 0; i < dataLen; ++i) sum += (int)data[i];

Vervolgend de modulo operatie met 256 uitvoeren:

sum %= 256;

En op het laatst te inverteren met de data lengte en dan de actie bij optellen:

sum ^= dataLen;

sum += actie;

Ook is er een buffer limiet op de RP6, ten grootte van 32 bytes (standaard). Omdat de informatie data van de protocol al 5 bytes in beslag neemt, zal de data lengte maximaal 27 bytes mogen zijn. Effectief 26 omdat voor strings veiligheid altijd data[dataLen] = 0 uitgevoerd moet worden zodat de data zelf altijd null-terminated is. Dit wordt gedaan bij het ontvangen, niet bij het versturen. Er mag dus maximaal 26 bytes gelezen en gestopt worden in ‘data’. Er mag pas begonnen worden met lezen wanneer de serial buffer minimaal 5 bytes bevat. Na het lezen van de data lengte moet worden gewacht totdat de serial buffer minimaal data lengte + 1 bytes bevat.

Voorbeeld van een buffer met een gestructureerde ‘Message’ met ‘hoi’ en actie nummer 5:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0xE6 | 0xC9 | 0x05 | 0x03 | 0x48 | 0x68 | 0x6F | 0x69 |

<=====startbytes=====><===actie==><=lengte==><==Check=><== (lengte)\*bytes = data =========>

Elke ‘Message’ moet in een keer (1 write) gestuurd worden, niet met meerdere writes: om te zorgen voor de beste performance.