Modbus Engine Dokumentáció

2015. március

[1. Mi az a ModbusEngine?](#h.ytqzw06q6xjt)

[1.1 A project](#h.wjt6akn652er)

[1.2 Felhasznált technológiák](#h.jm8nn8cfzpzg)

[1.3 Felhasznált library-k](#h.c0wbxrme0i7l)

[2. Project felépítés](#h.x4q4gfnvnmgc)

[2.1 Blokk diagram](#h.6nvctvpyk6fr)

[2.2 ModbusDriver](#h.fh015xr8jft1)

[2.3 TagSynchronizer](#h.bd64785ivs19)

[2.4 MonitorSynchronizer](#h.5wdlt3wkzahb)

[2.5 SQLDriver](#h.e4amk0vgiekh)

[2.6 MBPro fájl](#h.2vuzeqpgd1h5)

[3. Osztály struktúra](#h.4ot4o2ivubmd)

[3.1 ModbusDriver osztályai](#h.2zxqww4cqj6j)

[3.2 TagSynchronizer osztályai](#h.oqck05gwrzkx)

[3.3 MonitorSynchronizer osztályai](#h.dimlxkxqyn8)

[3.4 SQLDriver osztályai](#h.qybl3sbo19ac)

[3.5 Engine és MBPro osztályok](#h.taqheonb565w)

[3.6 Core osztályok](#h.pp887tkyqwog)

[3.7 main.cpp](#h.4hwu4xb6hdyk)

[4. Telepítés, használat](#h.fyimcgrborv1)

[4.1 Függőségek telepítése](#h.arc7ni7loa5g)

[4.2 Bináris futtatása](#h.i4r3owvsg6es)

[4.3 Forráskód fordítása](#h.clswtigm8qxq)

# 1. Mi az a ModbusEngine?

## 1.1 A project

A ModbusEngine tulajdonképpen nem más, mint egy – jelenleg linuxon – futtatható parancssori alkalmazás, mely modbus protokollon keresztül eléri a kívánt vezérlő eszközöket, belőlük adatot olvas és feléjük ír, azaz kétirányú kommunikációt folytat. A működésének specifikálásához egyetlen XML fájl elegendő, amiben definiálhatjuk az eszközöket, blokkokat, tageket.

## 1.2 Felhasznált technológiák

A ModbusEngine C++-ban íródott, a Qt Creator nevű IDE-ben. A Qt library-t nem, viszont a qmake programot felhasználtam a fordításhoz. A projekt forrása olyan szerkezetű, hogy könnyedén megnyitható bármelyik Qt Creator IDE-ben. Ehhez a forrás gyökerében található modbusengine.pro fájlt kell megnyitnunk a Creatorban.

**Fontos!**

**A project használja a C++11 szabvány által leírt osztályokat, módszereket is!**

A projekthez a MySQL adatbázis kezelőt választottam, ugyanis az alkalmazás a külvilággal teljes mértékben egy általa épített adatbázis segítségével kommunikál.

## 1.3 Felhasznált library-k

A standard C és C++ library-k mellett a project két külső library-t használ:

1. libmodbus – Modbus C library ( 3.0.5 )

2. libmysqlcppconn – C++ Connector Library for MySQL Databases ( 7.1.1.3 )

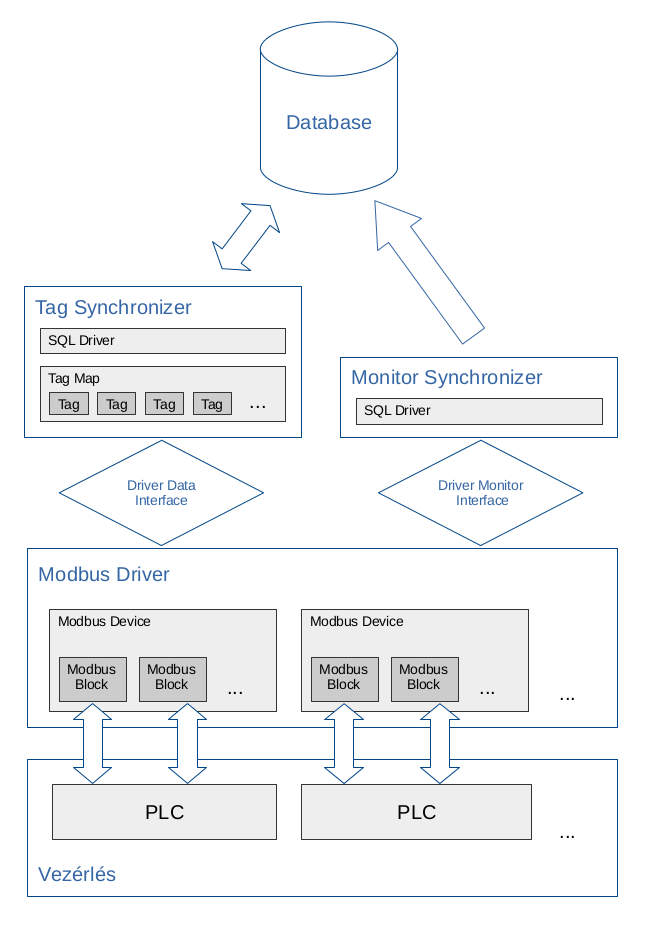
# 

# 

# 2. Project felépítés

A program rétegekből áll. Minden rétegnek megvan a maga feladata és a csatlakozási felületei. Ebben a fejezetben ezekről a rétegekről lesz szó kicsit részletesebben.

## 2.1 Blokk diagram



## 2.2 ModbusDriver

A ModbusDriver a legalsó szint, az ő feladata az, hogy elérje a PLC-ket vagy bármilyen Modbus kommunikációra képes eszközt. Három fő összetevőből épül fel a modul: **ModbusBlock**, **ModbusDevice** és **ModbusDriver**.

A **ModbusBlock** tárolja a Modbus kérdés specifikációját, az offset és count mezőt, az olvasási ciklusidőt, a hiba utáni várakozási időt, a Modbus hibaüzeneteket, az írási hiba utáni újrapróbálkozások számát és egyéb paramétereket. A ModbusBlock az eszközhöz tartozó kapcsolatot az őt tartalmazó ModbusDevice-tól kéri. Minden ModbusBlock külön szálon fut, így párhuzamosan képesek futni, felgyorsítva ezzel a kommunikációt. A ModbusBlock működése közben folyamatosan a megadott ciklusidővel olvas és írási flag hatására a kért adatot leírja az eszköz megfelelő regiszterébe.

A **ModbusDevice** tárolja a kapcsolódási paramétereket, az IP címet, a port számot, a Slave ID-t, a válaszadási timeout-ot, és a kapcsolódási timeout-ot. A ModbusDevice-on belül tároljuk a ModbusBlock-okat. A ModbusDevice tartalmazza a kapcsolatot is, de nem ő építi fel, hanem mindig az ún. Master Block, amit a program futás elején az első ModbusBlocknak osztunk ki.

Mind a ModbusBlock-ok, mind a ModbusDevice-ok rendelkeznek egyedi ID-vel. Egy ModbusDriver-en belül egy ModbusDevice ID csak egyszer fordulhat elő, ugyanígy egy ModbusDevice-on belül minden ModbusBlock ID is egyedi kell hogy legyen.

A **ModbusDriver** tartalmazza a ModbusDevice-okat, valamint hozzá vannak csatolva a külvilág felől elérhető interfészek. Két ilyen interfész van. Az egyik a Modbus kommunikációs adatok elérését biztosítja, ez a ModbusDriverDataInterface. A másik a monitorozáshoz és hibakezeléshez fontos ModbusDriverMonitorInterface, mellyel nyomon követhetjük az eszközök kapcsolatának állapotát és a blokkok működését.

A **ModbusDriverDataInterface** segítségével elérhetjük tehát a kommunikációs adatokat. Az interfész BIT, BYTE és WORD típusú TAG-ek olvasására és írására ad lehetőséget. A többi összetettebb adattípus mint pl. a DWORD ezekből az elemi addattípusokból származtatható.

A **ModbusDriverMonitorInterface** a ModbusDevice-ok és ModbusBlock-ok paramétereit és állapotait szolgáltatja a külvilág felé.

A ModbusDriver indulásakor a konfigurációs XML fájl alapján felépíti a ModbusDevice-okból és ModbusBlock-okból álló struktúrát.. Ezzel a modul sikeresen felépül.

## 2.3 TagSynchronizer

A TagSynchronizer modul a ModbusDriver felett helyezkedik el, hozzá a ModbusDriverDataInterface-en keresztül kapcsolódik. Feladata hogy a TAG absztrakciót elvégezze, azaz hogy az alacsony szintű alapvetően WORD-ös szervezésű Modbus memóriatérképet leképezze különböző típusú TAG-ekre, azokat a memóriában tárolja és megfelelő időközönként adatbázisba mentse, valamint az adatbázisba írt írandó értékeket lekérdezze és azokat továbbítsa a ModbusDriverDataInterface-en keresztül a ModbusDriver-nek.

**Jelenleg a következő TAG típusokat támogatja a rendszer:**

**BIT** – egy bites egész [0,1]

**BYTE** – 8 bites előjeles egész [-128,127]

**WORD** – 16 bites előjeles egész [-32768,32767]

**DWORD** – 32 bites előjeles egész [-2147483648,2147483647]

**REAL16** – 16 bites előjeles egész, osztva egész számmal lebegőpontossá tehető [-32768,32767]

**UBYTE** – 8 bites egész [0,255]

**UWORD** – 16 bites egész [0,65535]

**UDWORD** – 32 bites egész [0,4294967295]

A TagSynchronizer indulásakor létrehozza a számára szükséges adattáblákat az adatbázisban, a konfigurációs XML alapján felépíti a TAG-ek struktúráját, azokat kiírja a létrehozott táblába.

## 2.4 MonitorSynchronizer

A MonitorSynchronizer modul szintén a ModbusDriver felett helyezkedik el, azt a ModbusDriverMonitorInterface-en keresztül éri el. Feladata hogy induláskor adatbázisba kiírja a ModbusDevice-ok és ModbusBlock-ok paramétereit, valamint futás közben azoknak állapotait.

Induláskor a MonitorSynchronizer létrehozza a neki szükséges adatbázis táblákat, valamint feltölti azokat tartalommal.

## 2.5 SQLDriver

Az SQLDriver tulajdonképpen az adatbázis eléréséhez szükséges függvényeket és motort biztosítja.

## 2.6 MBPro fájl

Az MBPro fájl ír le egy ModbusEngine projectet. Ez egy szabványos XML fájl melynek a struktúrája a következő:

A gyökérelem az *<mbpro>* tag. Ezen belül négy alrész helyezkedik el:

*<project>* : projekt információk

*<db>* : adabázis konfiguráció

*<modbusdriver>* : modbus driver konfiguráció

*<taglist>* : tag lista konfiguráció

A *<project>* tag egyetlen al-taget tartalmaz, a *<name>* tag-et mely csupán egy információ a projekt nevéről, sehol sincs használva mint azonosító.

A *<db>* tag a következő al-tagekkel rendelkezik:

*<dbType>* : jelenleg mindegy mi, csak a mysql támogatott

*<dbUrl>* : az adatbázis elérési útvonala URL formátumban

*<dbPort>* : adatbázis szerver portja

*<dbName>* : adatbázis neve

*<dbUser>* : adatbázis felhasználónév

*<dbPass>* : adatbázis jelszó

A *<modbusdriver>* tag egy al-taggel rendelkezik a *<devices>* taggel. A *<devices>* tag tetszőleges számú *<device>* taget tartalmazhat. A *<device>* tag al-tagjei:

*<deviceId>* : egyedi device azonosító

*<ip>* : az eszköz IP címe

*<port>* : az eszköz portszáma

*<slaveId>* : az eszköz slave ID ja (Modbus specifikus)

*<responseTimeout>* : várakozási idő millisecundumban a válasz érkezésére

*<connectionTimeout>* : várakozási idő a kapcsolódás-kérelmi válaszra (ms)

*<blocks>* : device-on belüli blockok tömbje

A *<blocks>* tag tetszőleges számú *<block>* taget tartalmazhat. A *<block>* tag al-tagjei:

*<blockId>* : a blokk egyedi azonosítója

*<offset>* : a modbus kérdés offsetje

*<count>* : a modbus kérdés countja

*<cycleTime>* : kérdezési ciklusidő

*<retries>* : írási hiba utáni újrapróbálkozások száma

A *<taglist>* tag tetszőleges számú *<tag>* taget tartalmazhat. A *<tag>* tag definiálja a címzést, a címzett adattípust, az elvégzendő egyéb műveleteket. A *<tag>* tagnek nincsenek al-tagjei, az adatok attribútumként adandók meg. A *<tag>* tag kötelező attribútumai:

*name* : egyedi azonosító név

*deviceId* : címzett ModbusDevice azonosítója

*blockId* : címzett ModbusBlock azonosítója

*address* : word sorszáma

*type* : adattípus [bit,byte,word,dword,real16,ubyte,uword,udword]

A *<tag>* tag lehetséges attribútumai:

*subAddress* : bit típus esetén a bit száma, byte/ubyte esetén a byte száma

*divider* : real16 adattípusnál használt osztó [10,100,1000,10000]

*add* : típussal megegyező eltolás ( hozzáadás )

*multiple* : típussal megegyező skálázás ( szorzás )

*wordSwap* : dword/udword esetén alkalmazható word-swap

Két példa XML konfigurációs állomány található a projekt mappájában a *doc/example\_project\_files* mappában.

# 3. Osztály struktúra

Az alábbiakban ismertetem az alkalmazás osztály struktúráját.

## 3.1 ModbusDriver osztályai

A ModbusDriver modul három fő osztályból és két interfész osztályból áll. Mindegyik osztály részletesen kommentelve van a header fájlokban. Részletes információk ott találhatóak. A modul osztályai:

*ModbusBlock* : ModbusBlock-ot definiálja

*ModbusDevice* : ModbusDevice-ot definiálja

*ModbusDriver*: ModbusDriver-t definiálja

*ModbusDriverDataInterface* : ModbusDriverDataInterface-t definiálja

*ModbusDriverMonitorInterface* : ModbusDriverMonitorInterface-t definiálja

## 3.2 TagSynchronizer osztályai

A TagSynchronizer modul tartalmazza a TAG absztrakt definiícióját, a különböző típusú TAG-ek megvalósítását és magát a TagSynchronizer osztályt. Az összes Tag típus-osztály a Tag ősosztályból száramzatatott így közös interfésszel rendelkeznek. Az osztályok:

*Tag* : absztrakt Tag ősosztály

*BitTag* : Bit típusú Tag reprezentációja

*ByteTag* : Byte típusú Tag reprezentációja

*WordTag* : Word típusú Tag reprezentációja

*DWordTag* : DWord típusú Tag reprezentációja

*Real16Tag* : Real16 típusú Tag reprezentációja

*UByteTag* : UByte típusú Tag reprezentációja

*UWordTag* : UWord típusú Tag reprezentációja

*UDWordTag* : UDWord típusú Tag reprezentációja

*TagSynchronizer* : TagSynchronizer reprezentációja

## 3.3 MonitorSynchronizer osztályai

A MonitorSynchronizer modul csak a *MonitorSynchronizer* osztályt tartalmazza.

## 3.4 SQLDriver osztályai

Az SQLDriver modul négy fő osztályt tartalmaz:

*SQLDriver* :

az absztrakt SQLDriver definiíciója

*MySQLDriver* :

az SQLDriver ősosztályból leszármaztatott osztály a MySQL eléréséhez

*SQLDriverException* :

definiált „szabványos” SQL kivétel

*SQLResult* :

„szabványos” adatbázis lekérdezés válasz-objektumot definiáló osztály

## 3.5 Engine és MBPro osztályok

Két osztály van még mely a modulokon kívül helyezkedik el.

Az egyik az Engine osztály, aminek a feladata hogy összefogja és egy helyen kezelje az MBPro, a ModbusDriver, a TagSynchronizer és a MonitorSynchronizer osztályok egy-egy példányát. Az Engine olyan interfésszel rendelkezik, melynek segítségével egyszerűen indítható a teljes alkalmazás.

A másik osztály az MBPro osztály, mely a konfigurációs MBPro XML fájl egy programobjektumként reprezentált változata.

## 3.6 Core osztályok

A modulokon kívül találhatóak osztályok a *Core* könytárban is. Ezek:

*Conversion* :

konverziót támogató függvényeket tartalmazó osztály

*MBTCPMasterConnection* :

Modbus TCP Master kapcsolat objektum

*NetworkTester* :

ping függvényt tartalmazó osztály

*Thread* :

szálkezeléshez készült ősosztály

## 3.7 main.cpp

A *main.cpp* állomány tartalmazza a main függvényt, azaz a program belépési pontját. Először itt példányosítjuk az Engine-t, felépítjük azt, majd egy gyermek processzust hozunk létre, az ős processzust bezárjuk és már az új gyermek folyamatban indítjuk el az Engine-t.

# 

# 

# 4. Telepítés, használat

A ModbusEngine telepítése Linux rendszerre történhet. A telepítés tulajdonképpen csak a függőségek telepítését és a lefordított bináris másolását jelenti. A bináris másolása helyett lehetőség van a projectet forráskódból újrafordítani adott platformra.

**Figyelem!**

**Jelenleg a fordítás csak Linux rendszereken működik módosítások nélkül!**

## 4.1 Függőségek telepítése

A szükséges összetevők:

- Linux rendszer

- MySQL szerver

- 1.3 pontban ismertetett library-k

## 4.2 Bináris futtatása

A bináris egy parancssorból indítható alkalmazás. Egyetlen parancssori argumentuma van, a konfigurációs XML fájl elérési útvonala.

Mielőtt az alkalmazást elindítjuk győződjünk meg arról hogy az adatbázis szerver a megfelelő címen és porton valóban fut, illetve hogy létrehoztuk e az üres adatbázist a megfelelő néven. Ha ezek nem teljesülnek, az indítás sikertelen lesz.

## 4.3 Forráskód fordítása

A forráskód futtatásához szükségünk van a megfelelő fejlesztői környzetre. Ez a következő alkotókból áll:

- Linux rendszer

- gcc fordító, g++ fordító

- qmake és Qt library

- Qt Creator IDE

- 1.3 pontban ismertetett library-k, header fájlokkal együtt

A projekt src mappájában található modbusengine.pro fájlt nyissuk meg Qt Creatorban majd fordítsuk le a projectet. Ha minden jól ment, hiba nélkül fordul a kód.