

# Pistetietokanta

Prosessidata tallennetaan CPU yksikön pistetietokantaan. Teknisesti tietokanta on suuri avain/arvo -taulukko, jonka sijaitsee erillisen käyttöjärjestelmäprosessin varaamassa muistissa.

Tietokannassa säilytetään dataa, jonka halutaan säilyvän järjestelmän uudelleenkäynnistyksen ylitse, näyttää käyttöliittymägraafiikassa, tai vaikkapa siirtää PLC ohjelmalta toiselle.

Tietokanta tallennetaan laitteen sisäiselle flash muistille **/opt/slc/data/persistent.json** -tiedostoon. Mikäli kyseistä tiedostoa ei ole laisinkaan olemassa kun laite käynnistyy (tai oikeammin slcengine ohjelma), luodaan laitteeseen tyhjä pistetietokanta. Mikäli tiedosto on korruptoitunut, eli se on olemassa, mutta siinä olevaa JSON dataa ei pystytä tulkitsemaan oikein, koettaa järjestelmä ladata ensin varmuuskopion **persistent.bak.1** ja mikäli se ei onnistu, sen jälkeenkoetetaan ladata toinen varmuuskopio **persistent.bak.2**. Jos mitään näistä tiedostoista ei saada ladattua, käynnistetään SLC Engine virhetilassa, jossa tietokannan levyoperaatiot ovat estettyjä. Tämä tehdään siksi, että ei ylikirjoitettaisi vahingossa persistent.json tiedostoa tai sen varmuuskopioita. Näin pistetietokannalle voi koettaa tehdä manuaalisen palautusoperaation mikäli edes osa datasta on edelleen lukukelpoista.

Pistetietokannan datapisteet voidaan ryhmitellä hierarkisesti hakemistoihin hieman tiedostojen tapaan, jossa hakemiston erottimena toimii kauttamerkki (/). Näin ollen prosessin ABC pisteet TE01, TE02 ja TE03 voidaan nimetä esimerkiksi ABC/TE01, ABC/TE02 ja ABC/TE03. Tämän jälkeen prosessin ABC -tietokantapisteet on mahdollista hakea vaikkapa komennolla **Data.list ("ABC/\*", "")**. Toisaalta, myös tietokantaselain näyttää kaikki kyseisen prosessin pisteet käyttöliittymässä samassa hakemistossa.

Tietokantapisteen nimi voi koostua merkeistä **a..z, A..Z, 0..9, / ja \_**. Muita merkkejä ei voi käyttää, sillä pisteiden luontiin käytetty kutsu Data.create () epäonnistuu mikäli pisteen nimi sisältää kiellettyjä merkkejä.

Tietokantapisteitä on mahdollista käsitellä Plc ohjelmassa "Data" -kirjaston kutsuilla. Kirjastossa on kutsut joilla on mahdollista paitsi lukea ja kirjoittaa pisteitä, myös listata, luoda ja poistaa niitä.

Tietokannan datapiste koostuu niinsanotuista kentistä (eng. field) – joita kutsutaan myös tietueksi tai ominaisuuksiksi (properties). Kyse on kuitenkin samasta asiasta. Ja tietokantapisteen tieto onkin varastoitu juuri näihin kenttiin, esimerkiksi pisteen ABC/TE01 kenttä 'pv' voi sisältää lämpötilamittauksen varsinaisen mittausravon.

Kenttien nimiä koskevat samat nimeämissäännöt kuin itse pisteitä, paitsi että myös '/' -merkki on kielletty. Kentät voivat sisältää paitsi yksinkertaisia luku tai tekstiarvoja, mutta myös monimutkaisempia tietorakenteita, kuten taulukoita (object) tai listoja (array). Tällaiset monimutkaisia datarakenteita sisältävät pisteet on kuitenkin luotava n.s. mallin (schema) pohjalta.

Tietokantapisteillä voi siis olla myös ominaisuus jota kutsutaan malliksi. Se määrittää tarkalleen mitä kenttiä piste voi sisältää, ja millaista tietoa kukin näistä kentistä voi pitää sisällään. Kyse on samasta asiasta kuin tietokanta tai XML skeemojen kohdalla. Myös näitä skeemoja, eli malleja on mahdollista luoda Plc ohjelmissa "Data" kirjaston kutsuilla.

Tietokanta web käyttöliittymässä.

Kun käyttäjä luo uutta pistettä käyttöliittymän avulla, ja painaa "+ New" painiketta tietokantaeditorissa, häneltä kysytään avautuvassa dialogissa paitsi pisteen nimeä, myös sen tyyppiä. Tuo pisteen tyyppi tarkoittaa nimenomaan mihin malliin (eng. schema) piste sidotaan. Pisteiden mallit voivat esimerkiksi määrätä, että pisteessä on oltava kentät description, pv ja dataSource. Lisäksi malli määrittää mitä datatyyppiä kentät ovat.

Datapisteiden kenttien tyypit:

**boolean** (aliakset: bool)

Totuusarvo, eli voi saada arvon "true" tai "false"

**real** (aliakset: float, number)

Reaaliluku, eli tutummin desimaaliluku. Tarkkuus on aina 64-bit.

**integer** (aliakset: int)

Kokonaisluku. Monissa yhteyksissä Plc ohjelmissa tämä myös tätä tyyppiä käsitellään 64-bittisenä liukulukuna, mutta aina kun se tallennetaan pistetietokantaan, se muutetaan 32-bittiseksi kokonaisluvuksi.

**string** (aliakset: text)

Merkkijono, eli teksti. Tietotyyppillä on oikeastaan vain yksi huomion arvoinen rajoite; Se ei voi sisältää n.s. kaksois-hipsua, eli lainausmerkkiä (""). Sallittu enimmäiskoko on >4KBi (eli 4096 ASCII merkkiä).

**reference**

Viittaus toiseen tietokantapisteeseen. Oikeastaan kyse on teksti tyyppisestä datasta, ja tärkein ero string -tyyppiseen tietueeseen onkin siinä että sille voidaan näyttää erilainen käyttäjän syöteikkuna.

**array**

Lista muotoinen tietue. Listan ja taulukon suurin ero on siinä, että listalla on aina järjestys, ja listan tietueet löytyvät peräkkäisistä kokonaisluvulla osoitettavista soluista. Listan ensimmäinen solu on aina indeksissä 1, ja jos ensimmäinen solu poistetaan, muuttuu kaikkien jäljelle jäävien solujen indeksi yhdellä pienemmäksi. Lisäksi numerointi on aina juokseva, eikä välissä ole koskaan aukkoja.

Plc ohjelmissa array tyyppinen kenttä näkyy lua taulukkona, jolla on juokseva numeraalien indeksi. Lua tutoriaaleissa siihen viitataan array -tyyppisenä indeksointina.

**object**

Tämä datatyyppi on toisaalta melkotavalla samanlainen kuin lista, mutta numeraalisten indeksien sijasta sen soluihin osoitetaankin nimillä, hieman samaan tapaan kuin itse tietokantapisteiden kenttiin. Tämä aiheuttaa sen, että objektin soluilla ei ole mitään suurempaa sisäistä järjestystä, jossa yhden voisi sanoa tulevan ennen toista. Toisaalta, ne eivät myöskään vaihda paikkaansa kun jokin soluista poistetaan. Object tyyppisen kentän solut voivat olla nimeltään vaikkapa "maanantai", "tiistai" ja "keskiviikko".

Plc ohjelmissa object tyyppinen kenttä näkyy lua taulukkona, jonka soluilla on "string" tyyppiset indeksit, ja joita voidaan listata esimerkiksi pairs() -funktiolla.

Datapiste voidaan lukea Plc ohjelmassa funktioilla:

```
Data.get ("point_id")  
Data.getInt ("point_id")  
Data.getReal("point_id")  
Data.getString("point_id")
```

Funktiot toimivat samalla tavalla, erona on vain paluuarvon tyyppi. Data.get () voi palauttaa minkä tyyppisen arvon tahansa, ja sde on ainoa tapa lukea **array** tai **object** muotoinen kenttä. Data.getInt() palauttaa aina kokonaisluvun, Data.getReal() palauttaa aina desimaaliluvun, ja Data.getString () puolestaan merkkijonon - niinkun nimikin kertoo.

On tärkeätä huomata, että näiden kutsujen erot ovat nimenomaan virhetilanteissa! Mikäli luettava tietue on olemassa, ja sen tyyppi on määritelty skeemassa "real", palauttaa myös Data.get() liukuluvun. Data.getString () puolestaan muuttaa datan merkkijonoksi, ja palauttaa sen. Data.getInt () leikkaa desimaalit pois, ja palauttaa kokonaislukuosan. Data.getReal () taas pakottaa luvun liukuluvuksi (eli tässä tapauksessa ei tee sille mitään) ja palauttaa sitten arvon.

Tällä on merkitystä Plc ohjelmien yksinkertaisuuden ja virhesietoisuuden kanssa. Mikäli pyydetty tietue ei olekaan ohjelmoijan ennakoimaa tyyppiä, tai se on poistettu, palauttavat Data.getInt() ja Data.getReal() aina arvoksi 0. Data.getString () palauttaa virhetilanteessa tyhjän merkkijonon – mutta kuitenkin merkkijonon. Dat.get () puolestaan voi palauttaa arvon **nil**, mikäli pyydettyä tietokantapistettä ei ole olemassa.

Kun käyttää datatyyppiin sopivaa aliasta Data.get() -funktiolle ohjelmaa kirjoittaessa, voidaan työläästä tyyppitarkistukset jättää usein tekemättä ohjelman toimivuuden kärsimättä.

Pisteitä on mahdollista lukea Data.get() kutsulla enemmän kuin yksi! Pisteiden nimessä on mahdollista käyttää \* merkkiä n.s. jokerina, jolloin ensimmäisestä argumentista muodostuu n.s. maski. Kutsu palauttaa taulukkona kaikki sellaiset pisteet, joiden nimi sopii annettuun maskiin.

Datapiste kirjoitetaan Plc ohjelmassa:

```
Data.set ("point_id", value)
```

Kutsun käyttäminen sinänsä on hyvin suoraviivaista, esimerkiksi pisteen ABC/TE01 kenttä pv voidaan kirjoittaa arvoon 19.4 lua-komennolla

```
Data.set ("ABC/TE01.pv", 19.4)
```

Toisaalta, kutsua voidaan käyttää myös toisella tavalla, jossa yhteen pisteeseen on yhdellä kutsulla mahdollista kirjoittaa useampi kenttä:

```
Data.set ("ABC/TE01", {pv=19.4, commStatus=1})
```

Ylläoleva esimerkki vaikuttaa varmasti lua-kieltä tuntevalle melko selvältä, eli toisena argumenttina voidaan antaa yksittäisen lukuarvon tai merkkijonon sijasta taulukko, jonka alkiot kutsu kirjoittaa pisteen kenttien arvoiksi. Ylläolevassa esimerkissä pisteeseen kirjoitetaan pv kenttään 19.4 ja commStatus kenttään arvo 1.

# Tietokantapisteiden kentät

Lähtökohtaisesti ohjelmisto ei määrää sitä, minkä nimisiä ja minkä muotoisia kenttiä tietokantapisteeseen voi luoda. On kuitenkin olemassa joitakin kenttien nimiä, jotka on varattu ohjelmiston sisäisesti tiettyyn tarkoitukseen. Alla on näistä varatuista, erityis merkityksen omaavista kentistä.

Huomaa, että lista ei välttämättä ole täydellinen, ja jotkin kolmannen osapuolen/erityiskentät kirjastot voivat tehdä oletuksia, joita tässä ei ole mainittu.

## **description (string)**

Järjestelmä olettaa tämän kentän olevan pisteen vapaamuotoinen kuvausteksti.

## **pv (vaihtelee pistetyypeittäin)**

Järjestelmä olettaa tämän kentän olevan pisteen varsinainen datasisältö, eli oloarvo. Sen datatyyppi voi vaihdella sen mukaan, mikä piste on kyseessä. AV pisteellä se on real tyyppinen, kun taas HVAC kirjaston PID säätimellä se on taulukko, joka sisältää real -tyyppisiä arvoja (yhden per säätöporras).

## **priority (int)**

Ilmoittaa millä prioriteetilla pisteen oloarvo on kirjoitettu. Prioriteetit vastaavat BACnet standardin prioriteetteja, eli 16 on pienin (n.s. auto tila) prioriteetti, ja 8 vastaa käsiohjausta (manual tila). Suuremman prioriteetin kirjoitus hylätään, ja 16 on vähäisin, ja 1 suurin sallittu prioriteetti.

## **writable (int)**

Määrittää, voiko pisteeseen kirjoittaa BACnet väylän kautta. Arvo 1 tarkoittaa, että kirjoittaminen on sallittua.

## **dataSource (string)**

Tämä kenttä sisältää tiedon siitä, mistä pisteen oloarvo tulisi lukea. Tarkemmat kuvaukset kommunikaatio protokollien yhteydessä.

## **dataTarget (string)**

Tämä kenttä sisältää tiedon siitä, mihin pisteen oloarvo tulee kirjoittaa. Tarkemmat kuvaukset kommunikaatio protokollien yhteydessä.

## **commStatus (int)**

Ilmoittaa, onko piste online vai offline tilassa. Arvo 1 tarkoittaa online ja/tai OK tilaa, ja sitä pienemmän erilaisia mm. kommunikaatio protokollasta riippuvia häiriötiloja. Tämä liittyy esimerkiksi dataSource tai dataTarget kentissä määriteltyyn väylä- tai lähiverkko liikenteeseen.

## **commTxt (int)**

Teksti muotoinen kuvaus siitä, millainen häiriö pisteen kommunikaatiossa on. Mikäli pisteen kommunikaatio ei ole häiriöllä, tekstin tilisi sisältää "Online" -merkkijono (suositus).

### **unit (int)**

Pisteen BACnet yksikkö, numeraalinen arvo tulee BACnet standardista.

### **dispUnit (string)**

Tekstimuotoinen yksikkö joka näytetään joissakin grafiikka elementeissä (kaikki eivät tue).

### **lowLimit(number)**

Alaraja pv kentän numeeriselle arvolle, eli tämän pienempää numeerista arvoa ei ole mahdollista kirjoittaa pisteen pv kenttään.

### **highLimit (number)**

Yläaraja pv kentän numeeriselle arvolle, eli tämän suurempaa numeerista arvoa ei ole mahdollista kirjoittaa pisteen pv kenttään.

### **onDelay (number)**

Tämä kenttä luo viivettä siihen, kuinka nopeasti pisteen raw arvon nouseva reuna välittyy oloarvoon, eli tavallisesti vaikutus on sama kuin vetohidastuksella. Toimii vain silloin kun pisteen arvo luetaan tai kirjoitetaan jonkin kommunikaatorajapinnan kautta. Yksikkö on millisekunti.

### **offDelay (number)**

Tämä kenttä luo viivettä siihen, kuinka nopeasti pisteen raw arvon laskeva reuna välittyy oloarvoon, eli tavallisesti vaikutus on sama kuin päästöhidastuksella. Toimii vain silloin kun pisteen arvo luetaan tai kirjoitetaan jonkin kommunikaatorajapinnan kautta. Yksikkö on millisekunti.

### **curve (reference)**

HVAC kirjaston toteuttama ominaisuus. Toimii dataSource kentän yhteydessä, ja tähän voidaan antaa pistetunnus, joka viittaa HVAC kirjaston hvacCurve pisteeseen. Tämän jälkeen kyseistä hvacCurve-käyrää käytetään kun raw arvo skaalataan pisteen oloarvoksi (pv-kenttä).

## **Data -kirjasto: rajapinta pistetietokantaan**

### **Data.create ("pointId" [, "schema"], initialValues )**

Tällä funktiokutsulla luodaan uusia datapisteitä tietokantaan. Kutsu palauttaa *true* mikäli pisteen luominen onnistui, ja *nil* mikäli se epäonnistui. Mikäli samanniminen piste on jo olemassa, kutsu palauttaa virheen (eli *nil*) eikä muuta millään tavalla olemassa olevaa pistettä.

Uudet kentät olemassa oleviin pisteisiin luodaan Data.set () kutsulla.

pointId on luotavan pisteen nimi. Sallitut merkit ovat pcre syntaksilla ilmaistuna [a..zA..Z0..9/\_].

schema parametri on valinnainen, ja kertoo luotavan pisteen tyyppin, eli skeeman.

initialValues on taulukko, jossa voidaan antaa luotavan pisteen kenttien arvot, esimerkiksi halutut hälytysviiveet, kuvaus, tai muita tietoja.

```
-- Create new point, and attach it to schema 'AI'
```

```
Data.create("ioPoints/TK01TE10_AI", "AI", {description="Inlet air temperature"})
```

```
-- Create new point, do not attach it to any schema
```

```
Data.create("point/with/no/schema", {description="I has no schema"})
```

**Data.get ("pointId" [, "strFilter"] )**  
**Data.getInt ("pointId" [, "strFilter"] )**  
**Data.getInteger ("pointId" [, "strFilter"] )**  
**Data.getInteger ("pointId" [, "strFilter"] )**  
**Data.getReal ("pointId" [, "strFilter"] )**  
**Data.getNumber("pointId" [, "strFilter"] )**  
**Data.getString ("pointId" [, "strFilter"] )**

Tällä funktiokutsulla luetaan varsinaisten datapisteiden arvoja pistetietokannasta. Se palauttaa joko pyydetyn pisteen tai kentän arvon, tai *nil* mikäli arvoa ei löydy.

*Data.get()* funktion variaatiot pakottavat paluuarvon tiettyyn tyyppiin, joka voi yksinkertaistaa ohjelmaa myöhemmin. Perusmuotoinen *Data.get ()* voi palauttaa mitä datatyyppiä tahansa, ja palauttaa arvon *nil* mikäli haettua tietokantapistettä ja/tai kenttää ei ole olemassa. *Data.getString()* palauttaa tyhjän, 0 pituisen merkkijono, ja *Data.getInt()* – ja muut numeraaliset arvon palauttavat funktiot – antavat luvun 0 mikäli haluttua tietoa ei ole olemassa (tai se on väärää muotoa).

*pointId* parametri voi olla joko pisteen nimi – jolloin kutsu palauttaa koko pisteen taulukkona, tai viittaus johonkin pisteen kenttään, jolloin kutsu palauttaa vain tuon kyseisen kentän arvon.

Kyseinen parametri voi olla jopa wild card -merkkejä sisältävä suodatin, jolloin yksi *Data.get ()* kutsu palauttaa kaikki ne pisteet joihin suodatin täsmää. Tässä moodissa on mahdollista käyttää myös *strFilter* parametria, jolla voidaan tehdä suodatusta pisteen sisältämien kenttien perusteella.

*Data.get("ABC/TE01") -- Return whole datapoint as table (or nil on failure)*

*Data.get("ABC/\*") -- Get all points starting with 'ABC/'*

*Data.get("/\*", "(pv > 0)") -- Returns all points having pv greater than zero.*

### **Data.set ("pointId", value)**

Asettaa datapisteen arvon. Tällä menetelmällä voidaan asettaa joko yksittäisen kentä arvo, tai useamman kentän arvo yhdellä kutsulla. Tästä kutsusta ei ole olemassa useampaa variaatiota, vaan tietokantaan kirjoitettava arvo on muutettava haluttuun muotoon ennen kutsun suorittamista. *pointId* on merkkijono joka kertoo mihin tietokantapisteeseen halutaan kirjoittaa. Se voi olla joko pisteen nimi, tai sisältää pisteellä erotettuna myös tietokantapisteen kentän.

*value* on tietokantapisteeseen kirjoitettava arvo. Tässä on olemassa kaksi mahdollisuutta. Mikäli kirjoitetaan vain tiettyyn tietokantapisteen kenttään, tämä parametri on usein numero tai merkkijono. Toisaalta, mikäli kirjoitus kohdistuu koko tietokantapisteeseen, *value:n* tulee olla taulukko, joka kirjoitetaan pisteeseen niin, että taulukon rivi "pv" kirjoitetaan pisteen vastaavaan kenttään "pv". tällä tavalla voidaan kirjoittaa tietokantapisteen monta kenttää yhdellä kutsulla. Kutsu palauttaa true mikäli kirjoitus onnistui, tai nil mikäli se epäonnistui. Kutsu voi myös luoda uusia data-kenttiä pisteisiin, mikäli datapisteen skeema sen sallii.

*Data.set ("ABC/TE01.pv", 19.4)*

*Data.set ("ABC/TE01", {pv=19.4, commStatus=1, commTxt="Online"})*

### **Data.list ("pointIdFilter"[, "fieldFilter"])**

Tällä kirjastokutsulla on mahdollista listata tietokantapisteitä haluttujen ehtojen mukaisesti. Kutsu palauttaa onnistuessaan taulukon – tyhjän taulukon mikäli yhtään ehtoihin täsmäävää pistettä ei löytynyt, tai nil mikäli kutsu epäonnistui.

*pointFilterId* on pisteiden nimiin sovellettava suodatin, jossa voi käyttää wild card merkkeinä \* ja ? merkkejä.

*fieldFilter* on puolestaan lua kielinen vertailu, jossa voi testata esimerkiksi pisteen kenttien olemassaoloa ja/tai arvoja. Tälle lyhyelle skriptille annetaan paikallisina muuttujina kaikki testattavan pisteen kentät, joten se voi testata niitä hyvin helposti. Pisteeseen katsotaan läpäisseen testin, jos tämä skripti palauttaa jonkin muun arvon kuin *false* tai *nil*.

Huomaa, että tämä funktio palauttaa suodattimiin täsmäävien pisteiden nimet listana, ei pisteiden arvoja. *Data.get()* puolestaan, mikäli sitä kutsutaan suodattimien kanssa, palauttaa pisteiden arvot. Sen vuoksi suorituskyvyn kannalta tämä kutsu on huomattavasti nopeampi.

```
local t = Data.list ("ioPoints/TK01*") -- list all point starting with 'ioPoints/TK01'
local t = Data.list ("*") -- list all points
local t = Data.list ("*", "(pv > 0)") -- list all points with pv greater than zero
```

### **Data.remove("pointId", "fieldFilter")**

Poistaa tietokantapisteen, tai pisteitä. Palauttaa poistettujen pisteiden määrän numerona, tai *nil* jos kutsu epäonnistui.

*pointId* on poistettavan pisteen nimi, tai samanlainen \* ja ? merkkejä sisältävä nimisuodatin kuin esimerkiksi *Data.list()* kutsulla.

*fieldFilter* toimii kuten *Data.list()* kutsulla.

```
Data.remove ("ioPoints/TK01/TE01_AI") -- Remove single point
Data.remove ("ioPoints/TK01*") -- Remove all points beginning with 'ioPoints/TK01'
Data.remove ("*", "(dataSource)") -- Remove all points having dataSource field
```

### **Data.clone("pointSrc", "pointDst")**

Tällä kutsulla on mahdollista monistaa jo olemassa oleva tietokantapiste. Kutsu luo uuden pisteen nimellä *pointDst* joka on pisteen *pointSrc* täydellinen kopio kutsuhetkellä.

Kutsu palauttaa *true* jos monistaminen onnistui, tai *nil* mikäli se epäonnistui.

```
-- Creates new point called ' ioPoints/TK01/TE31 '
-- from point ' ioPoints/TK01/TE30 '
Data.clone ("ioPoints/TK01/TE30", "ioPoints/TK01/TE31")
```

### **Data.exists("pointId")**

Suorittaa testin onko *pointId* niminen piste tietokannassa. Funktio palauttaa *true* jos piste on olemassa, tai *false* mikäli sitä ei löydy.

```
if Data.exists ("ioPoints/TK01/TE10") then
    -- Point exists, do something
```



*else*

*-- Point does not exist, act properly*

*end*

### **Data.count ("pointIdFilter" [, "fieldFilter"])**

Laskee *pointIdFilter* ja valinnaiseen *fieldFilter* suodattimeen täsmäävien pisteiden määrän tietokannassa. Palauttaa numeron jos kutsu onnistui, ja nil jos se ei onnistunut. Huomaa, että se ei ole virhe jos tietokannasta ei löydy suodattimeen täsmääviä pisteitä, vaan kutsu palauttaa numeron 0 (nolla). Kutsu voi epäonnistua esimerkiksi siksi, että suodattimet sisältävät kiellettyjä merkkejä.

*-- Counts active alarms*

*Data.count("?", "(av and av > 0)" )*

---

Revision #2

Created 25 May 2022 07:15:34 by Severi Hiltunen

Updated 10 May 2023 10:25:34 by Severi Hiltunen