SLC

sisältää SLC enginen tehtävien hallitsemiseen tarvittavia funktioita. Sovellusohjelmien tarvitsee käyttää näitä vain harvoin.

Tärkeä huomio!
Kutsut jotka vaikuttavat plc-prosessin suoritusparametreihin tai hakevat tietoa niistä eivät tavallisesti toimi laisinkaan mikäli niitä kutsutaan Luan **coroutine** säikeen sisältä, koska tällainen suoritussäie näyttää Slc-kirjaston kutsujen mielestä toiselta lua virtuaalikoneelta (prosessi toisen prosessin sisällä). Näin ollen Slc-kirjaston kutsu ei tiedä mihin plc-prosessiin kutsu kohdistuu.

Slc.enableWatchdog ()
Slc.disableWatchdog ()
Ei argumentteja

Koko järjestelmän watchdog ominaisuuden ottaminen käyttöön, tai asettaminen pois päältä. Huomaa, että monilla laitealustoilla ei ole mahdollista kytkeä watchdog ajastinta pois päältä sen käynnistämisen jälkeen. Kun watchdog ajastin on käynnistetty Slc.enableWatchdog() kutsulla, slc engine huolehtii automaattisesti ajastimen ajoittaisesta resetoinnista (kutsutaan usein vahtikoiran syöttämiseksi) niin kauan on käynnissä, eikä sovellusohjelmanm tarvitse välittää asiasta. Oikea tapa käyttää näitä funktiota on esimerkiksi kutsua toista niistä yhden kerran järjestelmän käynnistyksen yhteydessä.

Paluuarvot
true jos onnistui, *nil* jos kohdattiin virhe.

Esimerkki:
```
 -- Set watchdog ON
    Slc.enableWatchdog ()
```

Slc.setWatchdogDelay (ms)
Slc.getWatchdogDelay ()
ms *Taskin vahtikoira-ajastimen viive millisekunteina*

Slc.error (strLevel, strMessage)

strLevel Virheen vakavuus: ”exception”, ”notify”, ”fault”, ”error”, ”critical”, ”fatal”

strMessage Virheen kuvausteksti

Ilmoittaa virhetilanteesta käyttäjälle, ja mikäli virhe on riittävän vakava, voi aiheuttaa myös PLC ohjelman siirtymisen virhe tilaan.

”error” taso aiheuttaa PLC ohjelman siirtymisen ”running” tilasta ”error” tilaan.

”critical” tai ”fatal” taso aiheuttaa PLC ohjelman suorituksen päättymisen, mikä yleensä aiheuttaa watchdog:n vuoksi koko laitteen uudelleen käynnistymisen.

Paluuarvot:
true mikäli onnistui, *nil* mikäli kohdattiin virhe (esimerkiksi virheellinen argumentti).

Esimerkki:
Slc.error(”error”, ”Houston, we had a problem”)

Slc.version ()
ei argumentteja

Lukee ja palauttaa järjestelmän ohjelmistoversion ja muita tietoja taulukkona. Taulukossa on kentät ”application” joka kertoo sovellusohjelman version (mm. monet plc kirjastot riippuvat tästä), ”kernel” joka kertoo käyttöjärjestelmän ytimen version ja ”plc” joka kertoo slc enginen binääritiedoston version.

Paluuarvot
taulukko jossa ohjelmistoversioiden tiedot.

Esimerkki:
print (”System kernel version: ” .. Slc.version().kernel)

Slc.createTask (strFile, strName, strMainCall, strSchMode, intDelay)

strFile lähdekooditiedoston koko polku

strName PLC tehtävän nimi

strMainCall tehtävän pääfunktion nimi

strSchMode Ajoitustila ”periocid” tai ”cyclic”

intDelay Ajoitusviive millisekunteina

Luo uuden PLC tehtävän annettujen parametrien perusteella. Tehtävä luodaan niin, että ensiksi tarkistetaan ovatko argumentit oikein, ja sen jälkeen käynnistetään uusi säie, ja viimeisessä vaiheessa ladataan ja käännetään annettu lähdekooditiedosto.

Paluuarvot:

true jos tehtävän luominen onnistuu, muuten *nil*.

Esimerkki:
Slc.createTask(”/opt/slc/prg/main.lua”, ”myTask”, ”main”, ”cyclic”, 500)

Slc.getSchedule ()
Slc.setSchedule (strMode)
strMode Haluttu ajoitustila, ”cyclic” tai ”periodic”

Funktioiden avulla voidaan lukea PLC ohjelman nykyinen ajoitustila, tai asettaa se halutuksi.

Slc.getSchedule palauttaa nykyisen tilan merkkijonona (”cyclic” tai ”periodic”).

Slc.setSchedule kutsulla se voidaan asettaa halutuksi,ja paluuarvo kertoo onnistuiko operaatio (true tai nil).

Esimerkki:

```
if Slc.getSchedule() ~= cyclic then
   Slc.setSchedule (”cyclic”)
end
```

Slc.getTiming ()
Slc.setTiming (intDelay)
intDelay Haluttu ajoitusviive

Funktioilla saadaan luettua PLC ohjelman nykyinen ajoitusviive, tai asetettua se halutuksi.

Slc.getTiming lukee viiveen ja palauttaa sen millisekunteina.

Slc.setTiming taas asettaa ohjelman viiveeksi halutun millisekuntimäärän. Palauttaa onnistuessa true, muutoin nil.

Slc.getTaskInfo ()
Slc.setName (strName)
strName Haluttu uusi PLC tehtävän nimi

Funktiolla getTaskInfo saadaan luettua PLC ohjelman tietoja, ja kutsu palauttaa taulukon jossa on kentät: ”name”, ”sourcefile”, ”callcounter”, ”runtime”, ”maincall”.

Name kenttä sisältää tehtävän nimen, sourcefile taas lähdekooditiedoston polun.

Callcounter kenttä on kokonaisluku joka kasvaa yhdellä joka kerran kun tehtävän pääfunktiota kutsutaan, eli se kertoo PLC ohjelman suorituskerrat.

Runtime kertoo kuinka kauan PLC ohjelman suorittaminen kestää, yksikkönä nanosekunti (PLC ohjelmien sisäinen ajoitus tehdään nanosekunteina).

SetName -kutsulla on mahdollista vaihtaa PLC ohjelman nimeä. Kutsu palauttaa true jos onnistuu, muutoin nil.

Esimerkki:
local info = Slc.getTaskInfo()
print (”task. ”..info.name..” running time: ”..info.runtime)

Slc.echo (strTxt)
strTxt *Tulostettava teksti*

Tulostaa tekstiä tai ohjelman tilaan liittyvää tietoa joka voidaan näyttää esimerkiksi grafiikka sivulla (mm. taskList -komponentti).

Paluuarvot
true mikäli onnistuu, muuten *nil*.

Esimerkki:
Slc.echo(n..” error on this run cycle”)

Slc.runRemote (strHost, strCmd)

strHost kohde laitteen IP-osoite tai DNS nimi.

strCmd Kohde koneessa suoritettava lua koodi

Tämän funktion avulla voidaan suorittaa lua komentoja toisessa SLC laitteessa (tcp portti 30001) ja lukea suoritetun lua-koodin paluuarvo takaisin ohjelmaan.

Funktiota voidaan käyttää hyvin monipuolisesti esimerkiksi tiedonsiirtomenetelmänä.

Huomioi! Tämä protokolla ei sisällä laisinkaan tietosuojaominaisuuksia! Jos tätä tiedonsiirtoprotokollaa käytetään julkisessa verkossa, se tulee suojata esimerkiksi SSH tunnelin avulla.

Esimerkki:
-- Lukee ulkolämpötilan LJH vakista
local te00 = Slc.runRemote (”192.168.0.201”, ”return Data.get(’ioPoints/TE00.pv’)”)

-- käynnistä poistopuhallin IV konehuoneessa

```
if not Slc.runRemote (”192.168.0.205”, ”return Data.set(’ioPoints/PK01/PF01.pv’, 1)”) then
    Slc.error (”fault”, ”Etäohjaus ei onnistu”)
```

```
end
```

Revision #5
Created 25 May 2022 09:25:20 by Severi Hiltunen
Updated 10 May 2023 10:25:34 by Severi Hiltunen