Artem Pirojenko

Olio-ohjelmointi LabVIEW-ympäristössä

Tyhjiömittausluokan ohjelmointi

Metropolia Ammattikorkeakoulu
Korkeakoulututkinto
Elektroniikka
Opinnäytetyö
29.12.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Artem Pirojenko Olio-ohjelmointi LabView-ympäristössä Tyhjiömittaus-luokan ohjelmointi 31 sivua + liite (45 s.) 29.12.2011
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Elektroniikan koulutusohjelma
Suuntautumisvaihtoehto	Elektroniikan suunnittelu
Ohjaajat	lehtori Janne Mäntykoski tutkija Alexandre Pirojenko

Tässä insinöörityössä on käsitelty olio-ohjelmointia graafisella ohjelmointikielellä *National Instrumentsin* LabVIEW-ympäristössä. Työ toteutettiin Helsingin Yliopiston Fysiikan laitoksella kiihdytinlaboratorion käyttöjärjestelmän kehityksen yhteydessä.

Koska hiukkaskiihdytinlaboratorion käyttöjärjestelmä on laaja ohjelma, sen ohjelmoiminen edellyttää toimintojen jakoa erillisiin osioihin ylläpidon helpottamiseksi. Koodin kokoonpanon tehostamiseksi osioiden on oltava modulaarisia. Luokkapohjainen olio-ohjelmointi tarjoaa tehokkaat työkalut modulaarisen koodin tuottamiseen.

Tässä työssä on esitetty tyhjiömittausantureiden luokan toteutus esimerkkinä olio-ohjelmoinnin tehokkuudesta. Esimerkkien avulla on kuvattu LabVIEW'n olio-ohjelmoinnin periaatteet, käyttökohteet, edut ja rajoitukset.

Työn tuloksena on ohjelmistomoduuli, joka on sellaisenaan valmis käyttöön esimerkiksi kiihdytinlaboratorion käyttöjärjestelmässä. Luokka integroidaan kiihdyttimen käyttöjärjestelmään sen käyttöönoton yhteydessä.

Avainsanat	LabView, olio-ohjelmointi, tyhjiömittaus, G-ohjelmointikieli, National Instruments

Metropolia

Abstract

Author Title Number of Pages Date	Artem Pirojenko Object-Oriented Programming in LabView Environment Vacuum-Measurement Class Programming 31 pages + appendix (45 p.) 29.12.2011
Degree	Bachelor of Engineering
Degree Programme	Electronics
Specialisation option	Electronics design
Instructors	Janne Mäntykoski, Senior Lecturer Alexandre Pirojenko, Researcher

This thesis deals with object-oriented programming with graphical programming language in National Instruments LabVIEW programming environment. The work has been carried out in the Physics Department of the University of Helsinki, in the Accelerator Laboratory, as a part development of an operating system.

Since the laboratory operating system is a wide program, its programming requires a division of the activities into separate partitions for future easy maintenance. To improve the composition of the partitions, the code must be modular. Class-based object-oriented programming offers powerful modular code generation tools.

As an example of object-oriented programming efficiency, this work presents the implementation of vacuum measuring sensor-class. The LabVIEW object-oriented programming principles, applications, advantages and limitations are described in examples.

The result of the work is a software module, which is ready to use, for example in the accelerator laboratory operating system. The vacuum measurement class will be integrated into the accelerator laboratory operating system during development.

Keywords	LabView, Vacuum measurement, G-programming language, object-oriented, National Instruments

Sisällys

Tiivistelmä

Abstract

Sisällys

1 **Johdanto** 1

2 **Ohjelmointi LabVIEW-ympäristössä** 2

2.1 LabVIEW'n ohjelmointikieli (G-kieli) 2
2.2 Olio-ohjelmionti G-kielellä 4

3 **Tyhjiömittaus** 6

4 **Tyhjiömittausluokan ohjelmointi** 8

4.1 Työssä käytetyt anturit ja ohjaimet 9
4.1.1 Tyhjiöanturit 9
4.1.2 Ohjaimet 10
4.1.3 Luokan data ja sisäiset muuttujat 16
4.1.4 Luokan aliohjelmat 17
4.1.5 Olion rajapinta 25

5 **Dokumentointi** 26

5.1 Taustaselostus (*Context Help*) 27
5.2 Lohkokaaviodokumentointi 28

6 **Yhteenveto** 30

Lähteet 31

Liite

Tyhjiömittausluokan lähdekoodi ja dokumentaatio

1 Johdanto

Nykypäivänä ohjelmointikieleltä vaaditaan toiminnallisuutta, nopeutta ja yksikertaisuutta, tämän takia ohjelmointikielet kehittyvät jatkuvasti kohti korkeampaa tasoa. LabVIEW'n G-ohjelmointikieli on näyttävä esimerkki hyvin korkean tason kielestä, joka kehittyy jatkuvasti tätäkin korkeammalle. Tässä insinöörityössä käsitellään olio-ohjelmointia graafisella G-ohjelmointikielellä National Instrumentsin LabVIEW-ohjelmointiympäristössä. Työ toteutettiin Helsingin Yliopiston Fysiikan laitoksella kiihdytinlaboratorion käyttöjärjestelmän kehityksen yhteydessä Helsingin Kumpulassa.

Olio-ohjelmointi sinänsä on hyvin tunnettu ja laajasti käytetty ohjelmointitapa eri ohjelmointikielissä. LabVIEW'n graafisella ohjelmointikielellä sen käyttö on suhteellisen uusi asia, mutta se osoittautui jo hyvin vahvaksi työkaluksi. Kiihdytinlaboratorion käyttöjärjestelmä on päätetty kehittää LabVIEW'n olio-ohjelmoinnilla, koska ohjelmointi on hyvin modulaarista, ja se on suunniteltu National Instrumentsin tuotteisiin, joita käytetään laboratoriossa laajasti. Tässä työssä on kuvattu LabVIEW'n olio-ohjelmoinnin periaatteet, edut ja rajoitukset.

Koska käyttöjärjestelmä on laaja ohjelma ja sitä kehitetään eri osastoissa, ohjelman on oltava mahdollisimman modulaarinen eli palapeliomainen, jolloin erilliset moduulit (ohjelmistokokonaisuudet) ovat helposti yhdistettävissä keskenään. Yhden sellaisen moduulin kehitys on tämän työn aihe.

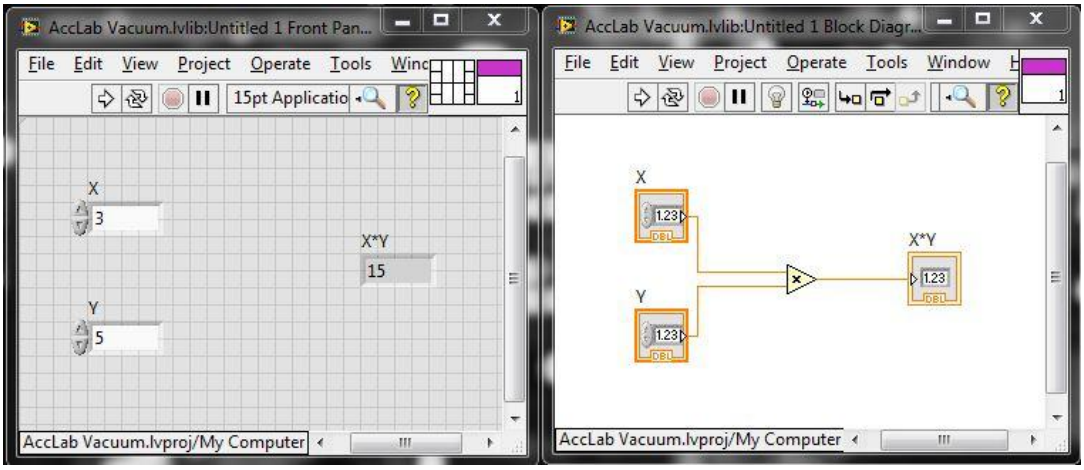
Aiheen moduuliksi on valittu tyhjiömittausantureiden luokka eli tyhjiömittausluokka. Luokasta on tehty pitkälle viimeistelty moduuli, jotta sen käyttöönotto olisi mahdollisimman vaivatonta toisen ohjelmoijan kannalta. Käyttäjän ei tarvitse lähteä tutkimaan luokan koodia käyttääkseen sitä omissa rakenteissa. Laboratorion laajan anturivalikoiman vuoksi, luokasta tehtiin mahdollisimman kattava, eli luokka kuvaa kaikkien käytössä olevien antureiden ja ohjaimien toimintaa. Käyttöjärjestelmässä luokan olioita lisätään aina, kun on tarve käsitellä paineantureita.

2 Ohjelmointi LabVIEW-ympäristössä

2.1 LabVIEW'n ohjelmointikieli (G-kieli)

Graafinen ohjelmointikieli on nimensä mukaan visuaalinen tai sommittelunomainen ohjelmointikieli. Toisin kuin tekstipohjaisissa kielissä, G-kielessä tekstin käyttö jää vähäiseksi. Tekstiä käytetään dokumentoinnissa ja sellaisissa tilanteissa, kun esimerkiksi yhtälön kirjoittaminen on helpompi kuin sen piirtäminen. LabVIEW on korkean tason ohjelmointiympäristö, joka on suunniteltu erityisesti laboratoriokäyttöön. Ohjelmointi perustuu niin kutsuttujen virtuaalityökalujen (ohjelmien) suunnitteluun ja keskinäiseen käyttöön (1 s. 1). Ohjelmointiympäristössä ohjelma koostu lohkokaavio-ikkunasta, jossa tapahtuu varsinainen ohjelmointi ja etupaneeli-ikkunasta, joka toimii käyttöliittymänä kontrolleineen ja indikaattoreineen (1 s. 4).

Ohjelmoidessa LabVIEW'ssa ohjelmoija yhdistää tarvittavat operandit/alifunktiot keskenään lohkokaavio-ikkunassa tietotyyppejä omaavilla johdoilla (kuva 1). Tarvittaessa rakenne sisällytetään erilaisiin silmukka-rakenteisiin.

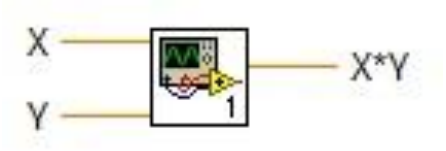


Kuva 1. Kertolaskuesimerkkiohjelma; vasemmalla etupaneeli-ikkuna ja oikealla lohkokaavio-ikkuna

Kuvassa 1 on toteutettu kertolaskuohjelma. Vasemmalla esitetyssä etupaneelissa on kaksi kontrolleria ja yksi indikaattori, jotka peilautuvat lohkokaavio-ikkunaan, jossa niitä

yhdistetään keskenään kertolaskualifunktion kautta. Kuva 1:n (s. 2) lohkokaaviosta voidaan havaita, miten jokaisen elementin sisääntulot sijoittuvat vasemmalle puolelle kuvaketta ja ulostulot oikealle. Helpottaakseen ohjelman hahmottamista sitä on tapana rakentaa vasemmalta oikealle tietovirran mukaan.

LabVIEW'ssa kehitetty ohjelma on käytettävissä sellaisenaan erillisenä ohjelmana ja myös muissa ohjelmissa aliohjelmana alifunktion tapaan. Kuvassa 2 näytetään, miltä kertolaskuohjelma näyttäisi toisessa ohjelmassa käytettynä.



Kuva 2. Kertolaskuohjelma lohkokaaviokuvakkeena

Isoja ohjelmia pystytään rakentamaan yhtenä koodipalana, mutta niistä tulee sotkuisia ja monimutkaisia. Ison ohjelman jako aliohjelmiin on hyvä tapa välttää sotkua ja parantaa järjestelmällisyyttä.

Ohjelmakoodin toteutus G-kielessä tapahtuu tietovirran mukaan, eli esim. alifunktio toteutuu jo silloin, kun sillä on kaikki sen tarvitsemat tiedot (1 s. 38). Sen jälkeen alifunktion tulos lähetetään seuraavalle, jossa dataa taas joutuu odottamaan, kunnes seuraavalla alifunktiolla on kaikki tiedot sisääntuloissa, toisin kuin esimerkiksi C++:ssa, jossa koodi toteutuu rivi riviltä riippumatta muuttujista. Toisin sanoen G-kielessä operandit tekevät tehtävänsä riippumatta naapuri-operandista silloin, kun mikään niiden sisääntuloista ei riipu rinnakkaisesta operandista. Tällä tavalla rinnakkaistoteutus ilmaisee itseään. Jos operandin sisääntulo on riippuvainen jostain toisesta operandista, ne toteutuvat peräkkäisjärjestyksessä.

Virheen havaitseminen G-ohjelmointikielessä on helppoa, koska koodi kääntyy jatkuvasti saman aikaan, kun koodia rakennetaan, eli ohjelma antaa ohjelmoijalle varoituksia tai ilmoittaa tapahtuneesta virheestä (1 s. 42). Kun koodi ei ole enää suoritettavissa ajaa-painike muuttuu rikkinäiseksi, ja se on merkki virheestä. Painamalla sitä ohjelma näyttää, missä tapahtui virhe.

2.2 Olio-ohjelmionti G-kielellä

Luokkaohjelmoinnin periaate, jolla tämä työ on toteutettu, on samanlainen kuin muissa tekstipohjaisissa kielissä, eli tehdään luokka, joka kuvaa tiettyä dataa ja metodiryhmää (1 s. 297). Seuraavat taulukot (taulukko 1 ja 2) esittävät luokkaesimerkkejä ja niiden oliota:

Taulukko 1. Luokkaesimerkit (2 s. 24)

Luokka:	**Auto**	**Henkilö**
Data:	Valmistaja Malli Vuosi Mittarilukema	Etunimi Sukunimi Syntymäaika Sukupuoli
Metodit:	Jarrujen tarkistus Renkaiden pyöritys Öljyn vaihto	Nimen tiedustelu Ikä tiedustelu

Luokkaesimerkit taulukosta näkee luokan sisällön. Datalohkossa on kaikki tiedot, joiden kanssa luokka on tekemisissä. Metodit-lohko sisältää käsittelymetodeja, jotka kuvaavat olion tietojen käsittelytapoja.

Taulukko 2. Olioesimerkit (2 s. 25)

Luokka:	**Auto**	**Henkilö**
Olio:	1964 Ford mustang 2004 Honda Accord	Artem Pirojenko Janne Mäntykoski

Taulukossa 2 on esitetty luokkien oliot, jotka sisältävät taulukko 1:n datalohkossa mainitut tiedot, joille voidaan soveltaa myös metodi-lohkon toiminnot.

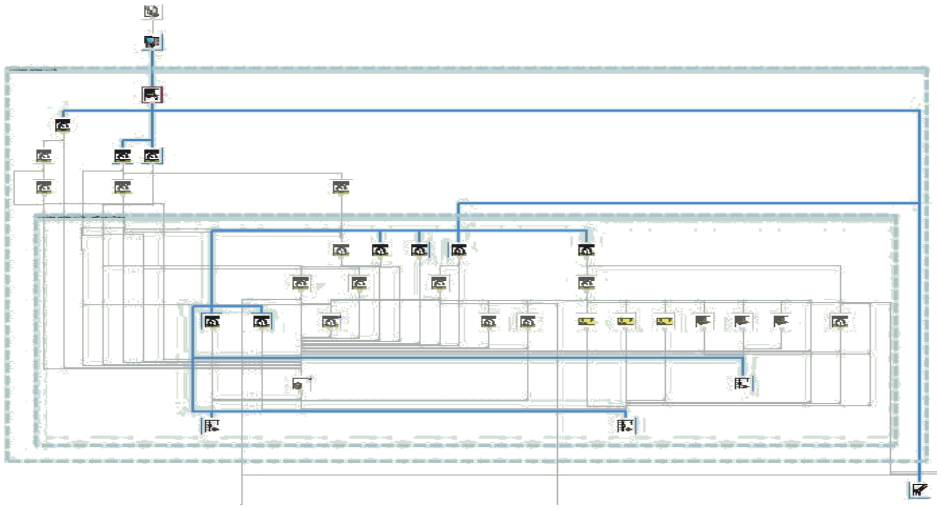
Luokkapohjaisen ohjelmoinnin periaate on, että kaikki luokan sisäiset tiedot (taulukko 1: data-lohko) ovat yksityisiä tietoja, eli niihin ei pysty vaikuttamaan mikään metodi-luokan ulkopuolelta. Hyvin määritelty ja dokumentoitu rajapinta olisi ainoa yhteys ympäröivään ohjelmaan tieto sisään ja vastaus ulos -periaatteella.

G-kielen olio-ohjelmointi, melkein samoin kuin tekstipohjaiset kielet, sisältää seuraavat hyödyt:

- tietojen suojaus enkapsuloinnilla
- virheen havaitseminen koodikehityksen aikana
- toiminnallisuus kätevän ohjelmointiympäristön ansiosta
- laajojen tietoklusterien eliminointi ohjelman pilkkomisella
- selkeiden suunnittelumallien toteutus (2 s. 23).

Luokkaohjelmoinnin pääajatuksena voidaan pitää seuraavia perustoimintoja: tietojen tallennus, tietojen lukeminen/kirjoittaminen ja tietojenkäsittely. Jokaisesta perustoiminnosta kehitetään oma metodi, joka toteuttaa kyseisen toiminnon tietojenkäsittelymenetelmän oliossa.

Olio-ohjelmoinnin rakenne muistuttaa Lego-palikoiden kokoamista. Varsinkin graafisessa ohjelmointikielessä ohjelmointi näyttää siltä. Lego-palikoiden sijaan G-kielessä esiintyy erilaiset metodit, joita kasataan yksinkertaisimmista alifunktioista ja tallennetaan yhtenä kokonaisuutena ohjelmakirjastoon. Samoja aliohjelmia/metodeja voidaan käyttää toisten metodien rakentamiseen, ja niistä myös tulee erillinen metodi toisenlaisin ominaisuuksiin ja toiminnoin, ja niitä taas käytetään sitä seuraavissa metodeissa.



Kuva 3. Esimerkki luokan metodien hierarkiasta visualisoituna (siniset johdot kuvaavat suorayhteyksiä ja harmaat välillisiä yhteyksiä)

Kuten kuvan 3 (s. 5) hierarkiasta voitiin havaita, että yksinkertaisimpiakin metodeja käytetään pitkin koodia paitsi suoraan niin myös välillisesti. Tästä voidaan päätellä, että G-kielen luokan metodihierarkia on joustava tai hajautettu.

Olio-ohjelmoinnin virhekäsittely tapahtuu luokan yhteisellä virherekisterillä. Kun luodaan luokan metodeja sen rakenteessa, on oletuksena kaksi johtoa ja niiden sisään ja ulos -tulot. Yksi niistä on datajohto, joka sisältää datatiedot. Toinen puolestaan on virhe/varoitusrekisteri, joka seuraa kaikkien luokan metodien toimintaa. Jos jossain metodissa ajon aikana tapahtuu toimintahäiriö, joka ei ole johtunut sen rakenteellisista puutteista (rakenteelliset puutteet korjataan käännös/kehitysvaiheessa), kyseisen metodin jälkeen suoritusjonossa olevat metodit saavat tiedon siitä, ja niillä on aina toimintatila tällaisia tapauksia varten. Tämän ansiosta koodiin voidaan lisätä ennaltaehkäisymetodit korjaaman mahdolliset virheet.

3 Tyhjiömittaus

Kiihdyttimessä hiukkasten kiihdyttäminen ja erilaisten kokeiden tekeminen tapahtuu tyhjiökammioissa, eli koko kiihdyttimen järjestelmässä koko ajan ylläpidetään tyhjiötä.
Tyhjiö on kaasun tai nesteen tila, jossa paine on pienempi kuin maan pinnalla vaikuttava ilmapaine, eli mitä harvemmin kaasua on jossain kammiossa, sitä korkeampi tyhjiö siellä on. Tyhjiö saadaan aikaiseksi pumpuilla, jotka jakautuvat eri toiminta-alueisiin: esityhjiössä omat pumput ja korkeimmissa tyhjiöarvoissa lisätään prosessiin korkeatyhjiöpumput. Tyhjiötä seurataan jatkuvasti antureiden avulla. Anturien mittausarvot vuorollaan välitetään tietokoneelle ja käsitellään kiihdyttimen käyttö-/ohjausjärjestelmää varten.

Tyhjiömittaus on fysikaalisten suureiden mittaus, jotka luonnehtivat tyhjiötä harvana kaasutilana ja määrittävät tyhjiön saamisen ja ylläpitämisen prosessit. Tyhjiö tai vakuumi (lat. *vacuum* eli tyhjä tila) on paineen tila, jossa se on pienempi kuin vertailupaine.

Paine on aineen nestemäistä ja kaasumaista olomuotoa koskeva suure, joka määritellään pintaa vastaan kohtisuoraan vaikuttavan voiman F ja saman pinnan alan A osamääränä:

$$p = F/A$$

missä

p	on paine
F	on voima ja
A	on pinta-ala

Paineen SI-järjestelmän mukainen yksikkö on *pascal* (Pa). Yksi *pascal* on paine, jonka yhden *newtonin* suuruinen voima tasaisesti jakautuneena aiheuttaa yhden neliömetrin pinta-alalle (1 Pa = 1 N/m^2) (3 s. 25). Työssä käytetään myös SI-järjestelmän ulkopuolisia suureita kuten torr (1 torr = 1 mmHg = 133.322 Pa).

Paineen mittaaminen on aina paine-eron mittaamista. Korkean vakuumin mittauksissa vertauspaineena käytetään absoluuttista tyhjiötä eli 0 Pa. Periaatteessa mitataan molekyylien määrä kaasussa.

Anturit, joita on tutkittu tämän työn yhteydessä, noudattivat seuraavia toimintaperiaatteita:

Kylmä-/kuuma-katodi (*cold/hot cathode*):
Ionisaatiomittareissa seurataan ionisoidun kaasun ionien määrä.

Pirani lämmönjohtavuus (*Pirani gauge*):
Lämmönjohtavuusmittareissa seurataan lämpöelementtin lämmönmenetystä, eli mitä harvempi kaasu on, sitä vähemmän siinä on lämpöä siirtäviä molekyylejä.

Kapasitanssi (*Capacitance*):
Kapasitanssimittareissa seurataan väliaineen suhteellista permittiivisyyttä, eli kuinka hyvin väliaine johtaa sähköä.

Termopari (*Thermopare*):
Samoin kuin *Pirani*-mittareissa termoparimittareissa seurataan lämmönjohtavuutta. Mutta tässä tapauksessa lämpöelementtiin syötetään vakiovirta, ja termoparilla seurataan sen lämpötilaa.

4 Tyhjiömittausluokan ohjelmointi

Projektiluokan kehittelyssä panostettiin sen selkeyteen ja yksikertaisuuteen. Luokkaan kuuluville aliohjelmille on annettu mahdollisimman kuvaavat nimet. Pohdittiin tarkoin, mitkä tiedot kuuluvat luokkaan ja mitkä jäävät esimerkkikoodin osioon. Luokan dokumentointia on pohdittu tarkasti, sillä modulaarisen ohjelmoinnin kannalta dokumentointi on hyvin tärkeä osa-alue, koska nimenomaan dokumentointi selvittää ohjelman tarkoitusta ja toiminnallisuutta tulevalle käyttäjälle (ks. kappale 5 dokumentointi).

Tyhjiömittausluokka nimensä mukaisesti kuvaa tyhjiömittauksia. Tämän takia ensisijaisesti selvitettiin, mitä kaikkea sen kuuluisi sisältää kattaakseen toiminnallisuuden ja tarkoituksen. Aluksi selvitettiin, mitä antureita on käytössä laboratoriossa ja mitä antureita suunnitellaan käyttöön otettavaksi lähiaikoina. Antureiden jälkeen katsottiin niiden ohjaimet, jotka välittävät anturin lukemat eteenpäin.

Ohjaimen ja tietokoneen väliin tulevia tiedonkeruulaitteita ei ruvettu tutkimaan tarkasti, sillä tiedonkeruulaitteet voivat muuttua, eikä se ole luokan kehityksen kannalta oleellista. Testausvaiheessa käytettiin National Instrumentsin tiedonkeruulaitetta USB-6009 ja Measurement & Automation -ajuriohjelmaa. Seuraavaksi kartoitettiin luokan datatiedot eli tiedot, jotka ovat yhteisiä luokan olioiden kannalta, ja viimeiseksi kartoitettiin metodit, jotka kuvaavat tietojen käsittelyä.

Luokan olion askeltoiminnallisuus on seuraava: mittauksissa käyttäjä määrittää nimen anturille, käytössä olevan signaalilähteen, ohjaintyypin, näytteenottojen määrän ja tietoyksikön. Ohjelma laskee ja palauttaa näiden tietojen pohjalta ohjaimen tila-arvon ja paineen lukuna.

4.1 Työssä käytetyt anturit ja ohjaimet

Tyhjiötä mitataan eri toimintaperiaatteilla toimivilla antureilla, antureiden ja tietokoneen väliin tulee ohjain, joka muokkaa signaalia tietokoneen ymmärtämään muotoon, tässä tapauksessa 0 - 10 V tasajännitettä, missä 0 V on hyvä tyhjiö, ja 10 V on tavallinen ilmapaine. Ohjaimen lähettämä signaali välitetään tietokoneelle tiedonkeruulaitteen kautta, kyseisen luokan testausvaiheessa käytettiin National Instrumentsin USB-6009 tiedonkeruulaitetta.

4.1.1 Tyhjiöanturit

Kiihdyttimen järjestelmässä käytetään monenlaisia tyhjiöantureita, jotka eroavat toiminta/mittausperiaatteilta ja mittausalueiltaan. Taulukossa 3 listataan kaikki mahdolliset anturit, joihin ohjaimien mukaan kyseinen tyhjiömittausluokka soveltuu:

Taulukko 3. Tyhjiömittausluokan anturit (*HVSensor.lvclass*)

Valmistaja	**Malli**	**Toimintaperiaate**	**Toimintaalue (Torr)**	**Ohjain**
MKS Instruments	Series 421, 423	Kylmäkatodi	1×10^{-11}-1×10^{-2}	HPS 937A
MKS Instruments	Series 315, 345	Pirani(Lämmönjohtavuus)	5×10^{-4}-$4 \times 10^{+2}$	HPS 937A
MKS Instruments	Series 317	Pirani(Lämmönjohtavuus)	1.0×10^{-3}-$1.0 \times 10^{+3}$	HPS 937A
MKS Instruments	622A, 626A, 722A	Kapasitanssi	1.0×10^{-2}-$1.0 \times 10^{+1}$	HPS 937A
InstruTech	CVG-101	Pirani (Lämmönjohtavuus)	1.00×10^{-4}-1000	VGC-301A
Granville Phillips	275	Lämmönjohtavuus	1×10^{-4}-1×10^{-1}	VGC-301A
InstruTech	CVG 101	Lämmönjohtavuus	1×10^{-4}-1000	IGM-402
InstruTech	IGM 402	Kuumakatodi	1×10^{-9}-5×10^{-2}	IGM-402

Työssä kuitenkin panostettiin enemmän niihin antureihin, jotka ovat käytössä Helsingin yliopiston laboratoriossa.

4.1.2 Ohjaimet

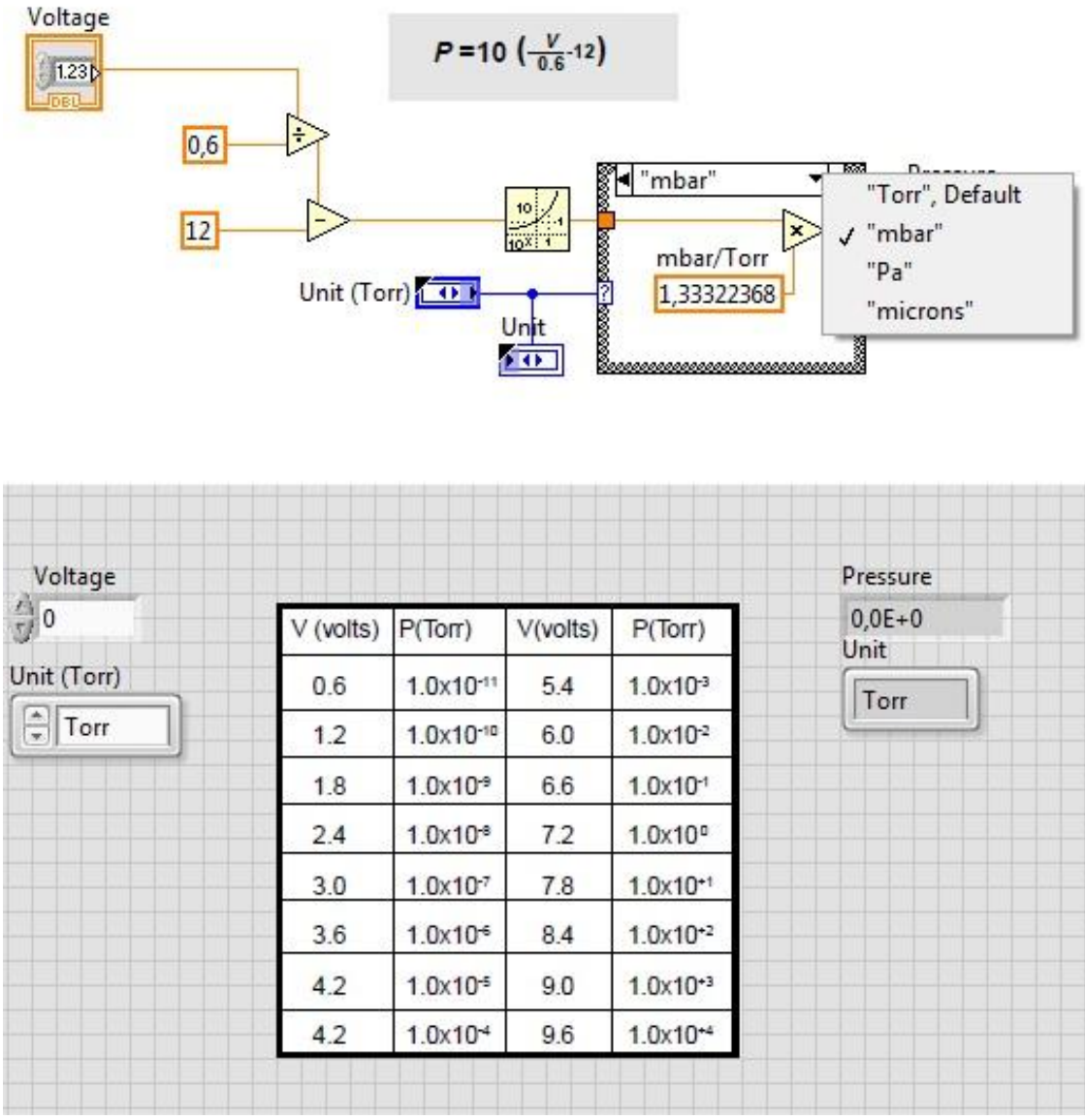
Taulukossa 4 esitetään kiihdytinlaboratorion kolme tyhjiöanturiohjainta ja niiden valmistajat seuravasti:

Taulukko 4. Tyhjiöanturiohjaimet

Ohjain	Valmistaja
HPS 937A	MKS Instruments
VGC-301A	InstruTech
IGM-402	InstruTech

Laboratoriossa on käytössä kolme varsinaista ohjainta (taulukko 4), jotka lukevat taulukossa 3 (s. 9) mainittuja antureita. Yhteistä niille on se, että kun mikä tahansa antureista on kytketty, ulostulossa on jännite 0 - 10 V. Se on juuri sopiva, sillä yleensä käytetyt tiedonkeruulaitteet toimivat nimenomaan sillä alueella. Lisäksi luokkaan on kuvattu tilanne, jossa ei tiedetä, mitä on kytketty, tai ohjain ei ole valmiina listassa, ja silloin ohjelma palauttaa lukemansa jännitteen arvon muokkaamattomana. Tämä mahdollistaa toiminnan erikoisemmissakin tilanteissa.

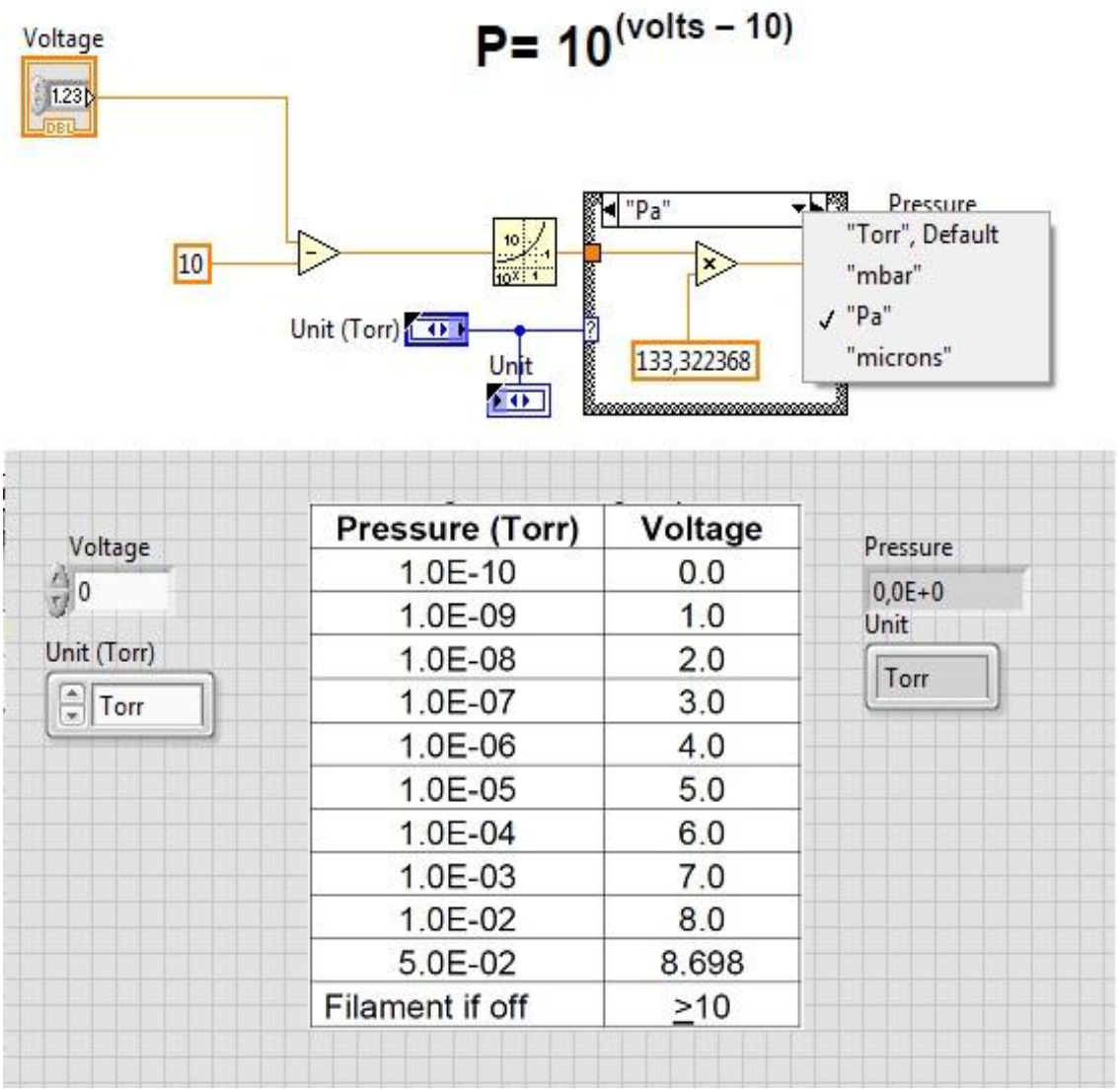
Aikaan saatiin neljä tilannetta, joista yhdessä ei tapahdu mitään, vaan tiedonkeruulaitteen sisääntulon jännitearvo lähetetään eteenpäin muokkaamatta. Tämän lisäksi saatiin kolme tilannetta, joissa tapahtuu jännitteestä paineeksi muunnos ja yksikkövalinta. Seuraavissa kuvissa 4, 5, 6 (s. 11, 12, 13) on esitetty kolme tilannetta G-kielellä toteutettuna. Kuvat on muokattu sillä tavalla, että lohkokaavio on ylempänä valkoisella alustalla ja alimpana, harmaalla, ruudullisella alustalla on kyseisen aliohjelman käyttöliittymä.



V (volts)	P(Torr)	V(volts)	P(Torr)
0.6	$1.0x10^{-11}$	5.4	$1.0x10^{-3}$
1.2	$1.0x10^{-10}$	6.0	$1.0x10^{-2}$
1.8	$1.0x10^{-9}$	6.6	$1.0x10^{-1}$
2.4	$1.0x10^{-8}$	7.2	$1.0x10^{0}$
3.0	$1.0x10^{-7}$	7.8	$1.0x10^{+1}$
3.6	$1.0x10^{-6}$	8.4	$1.0x10^{+2}$
4.2	$1.0x10^{-5}$	9.0	$1.0x10^{+3}$
4.2	$1.0x10^{-4}$	9.6	$1.0x10^{+4}$

Kuva 4. Aliohjelma *sub Voltage to Pressure (HPS 937A).vi* muuntaa ja palauttaa HPS 937A -ohjaimen painearvon; *Case*-rakenteeseen on listattu yksikkömuunnokset

Kuvassa 4 on esitetty, miten G-kielessä on toteutettu HPS937A-ohjaimen jännitteestä paineeseen (Torr) muunnos $(P = 10^{(V/0,6-12)})$ (4 s. 23). *Case*-rakenteessa on lueteltu muut yksikkömuunnokset Torr - mbar, Torr - Pa ja Torr - microns. Samanlainen *case*-valinta on kopioitu kaikkiin kolmeen aliohjelmaan, jotka kuvaavat ohjaimien painemuunnokset.

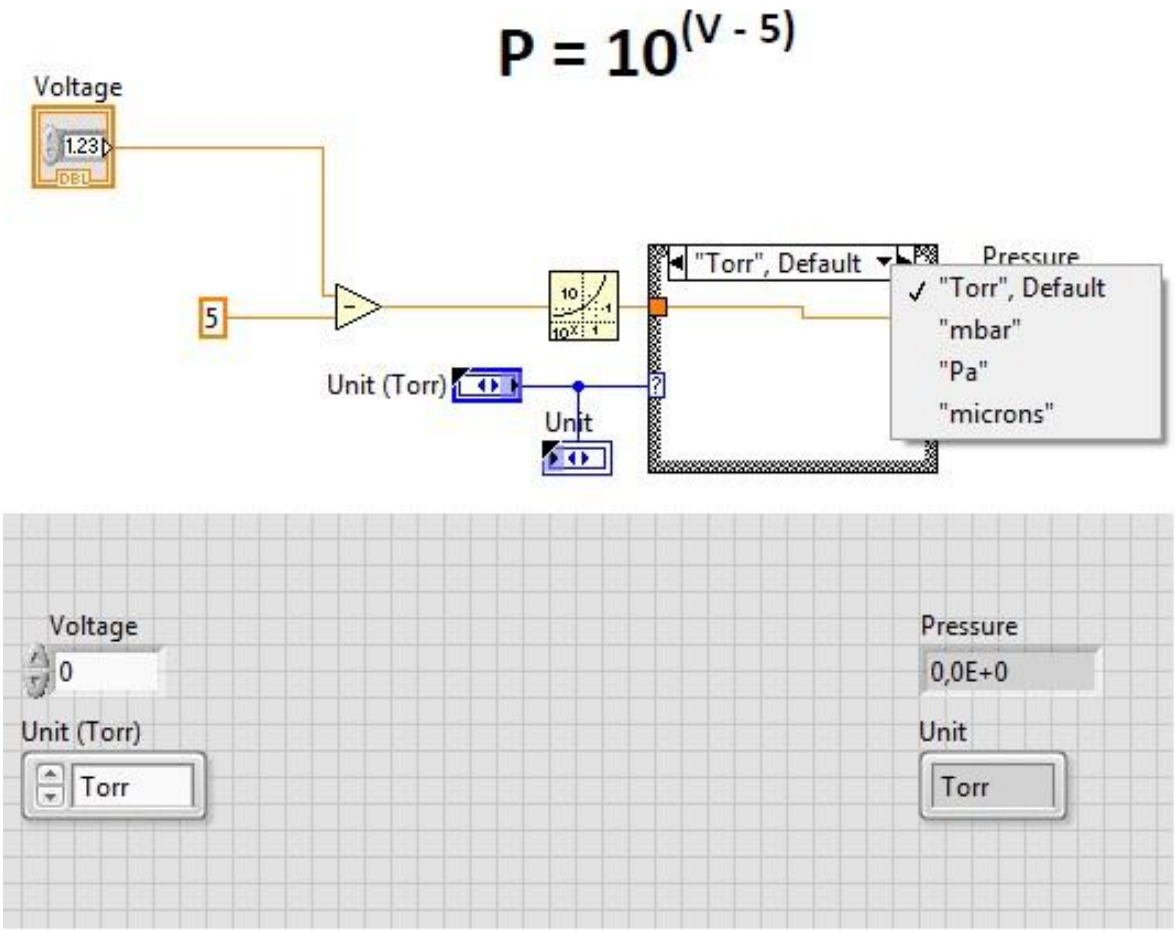


Pressure (Torr)	Voltage
1.0E-10	0.0
1.0E-09	1.0
1.0E-08	2.0
1.0E-07	3.0
1.0E-06	4.0
1.0E-05	5.0
1.0E-04	6.0
1.0E-03	7.0
1.0E-02	8.0
5.0E-02	8.698
Filament if off	≥10

Kuva 5. Aliohjelma *sub Voltage to Pressure (IGM-402).vi* muuntaa ja palauttaa IGM 402 -ohjaimen painearvon; *Case*-rakenteeseen on listattu yksikkömuunnokset

Käyttääkseen jokaiseen ohjaimen kuvaukseen samanlaista yksikkömuunnosvalintarakennetta, ohjaimien ulostulot muunnettiin ensin *torr*-yksikköön, koska valintarakenteen yksikkömuunnoksien alkuarvo on aina *torr*. IGM 402 -muunnosaliohjelman (kuva 5) muunnosyhtälö on seuraava:

$$P = 10^{(V-10)}\ [\text{torr}]$$ (5 s. 40)



Kuva 6. Aliohjelma *sub Voltage to Pressure (VGC-301A).vi* muuntaa ja palauttaa VGC 301A -ohjaimen painearvon; *Case*-rakenteeseen on listattu yksikkömuunnokset

VGC 301A muunnosaliohjelman (kuva 6) muunnosyhtälö on seuraava:

$$P=10^{(V-5)}\ [torr] \quad (6\ s.\ 35)$$

Edellisen kolmen kuvan 4, 5, 6 (s. 11, 12, 13) perusteella voitiin havaita, että aliohjelmat, jotka eivät erotu toisistaan kovin paljon voidaan kopioida ja monistaa niin paljon kuin on tarvetta. Jos luokan kirjastoon halutaan lisätä vielä yksi ohjain tai ohjaimen tila, kopioidaan jokin edeltäjistä ja muokataan sitä uusien ominaisuuksien mukaan.

(Kaikki luokan aliohjelmien ja esimerkkiohjelmien selostustiedostot ja koodit, ks. liite.)

Luokan rakenne

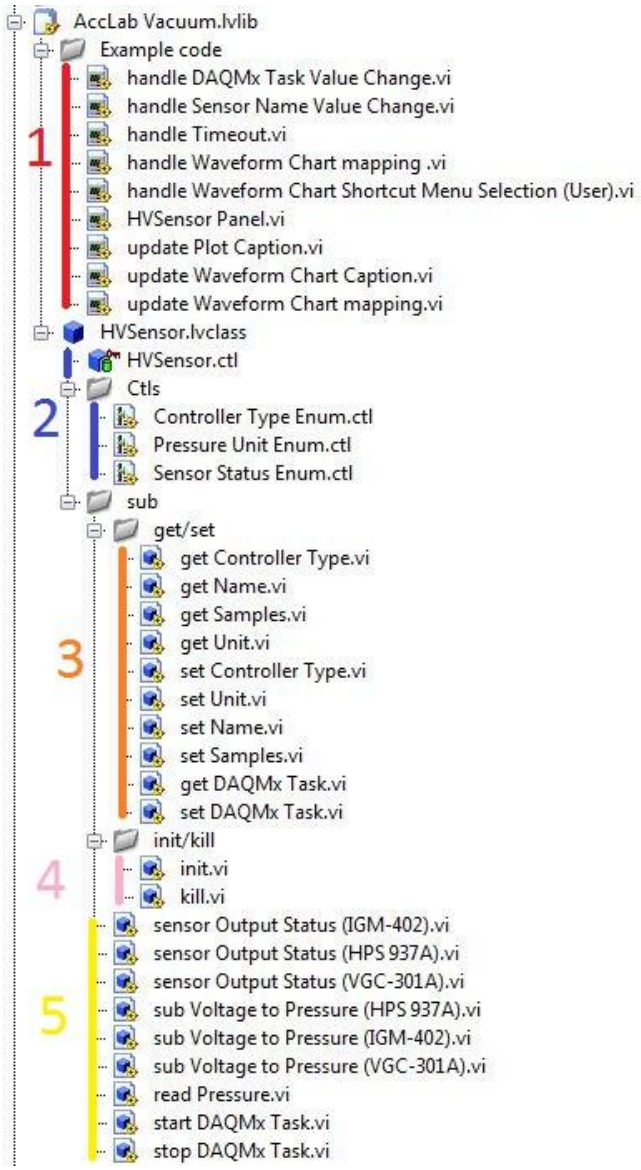
Luokan rakenne voidaan jakaa kahteen pääosioon: tiedonhallinta, jossa määritellään luokan sisäiset tiedot, sekä aliohjelmat, jotka sisältävät kaikki aliohjelmat, joilla on pääsy luokan sisäisiin tietoihin. Ne kaikki luodaan osana luokkaa, eli ohjelmat on valmiiksi mukautettu luokan sisäisten tietojen mukaan. Paitsi **.ctl*-tiedostot, joilla ei ole lohkokaaviota, vaan ne sisältävät tietoja luokan instansseista. Kuva 7 esittää yksinkertaisen luokan rakenteen:

Kuva 7. Yksinkertaisen luokan rakenne-esimerkki (2 s. 26)

Edellä esitetyssä rakenteessa (kuva 7) on kolme tiedostoa: *Vehicle.ctl* on luokan datatiedosto, joka sisältää luokan yksittäisdataa; *Start*- ja *Stop* -*Vehicle.vi* ovat luokan metodeja, eli ne kuvaavat luokan toimintoja.

Koska isoissa projekteissa toimintoja on monenlaisia, ja paljon, on hyvä saada ne järjestykseen. Aliohjelmia voidaan jakaa toimintatarkoituksen mukaan.

Kuvassa 8 (s. 15) esitetään tyhjiömittausluokan rakenne. Siitä voidaan nähdä, miten aliohjelmien jako ryhmiin on toteutettu kyseisessä projektissa. Väreillä ja numeroilla 1, 2, 3, 4 ja 5 erotetaan erilaiset aliohjelmaryhmät.

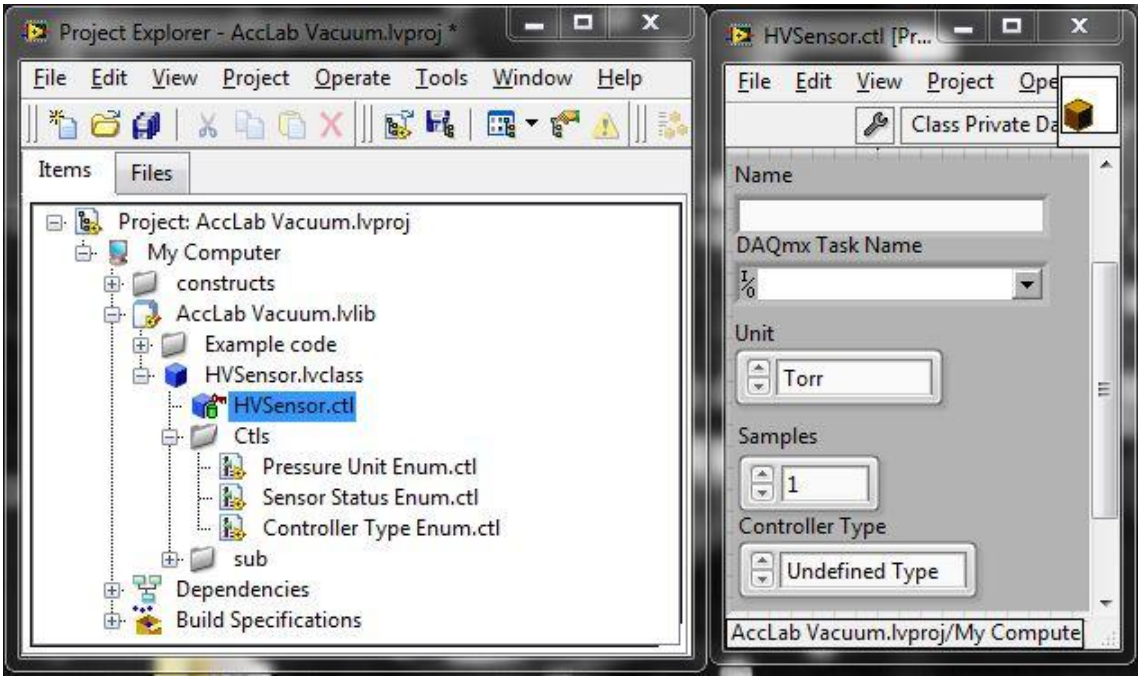


Kuva 8. Tyhjiömittausluokan rakenne; № 1 aliohjelmat ovat esimerkkiohjelmat; № 2 aliohjelmat ovat luokan data; № 3 aliohjelmat ovat luku- ja kirjoitus –ohjelmat; № 4 aliohjelmat ovat alustus- ja lopetus -ohjelmat ja № 5 aliohjelmat ovat tietojenkäsittelyohjelmat

Luokan rakennekuvassa (kuva 8) numero 1 sisältää esimerkkiohjelmat. Ne on kehitetty käyttäjää varten, eli ne antavat kuvan seuraavalle kehittäjälle siitä, miten luokan metodeja pystytään käyttämään. Numero 2 sisältää datatiedoston ja enumeraattorit/luettelot, joita on käytetty datatiedostossa. Numero 3 sisältää luku- ja kirjoitusohjelmat, joiden toiminnot rajautuvat luokan tietojen purkuun/lukemiseen ja kirjoittamiseen. Numero 4 sisältää alustusohjelmat, joiden tarkoitus on luoda ja lopettaa oliot. Numero 5 sisältää tietojenkäsittelyohjelmat ja ohjelmat, joita ei pystytä lajittelemaan yllämainittuihin ryhmiin.

4.1.3 Luokan data ja sisäiset muuttujat

Luokan sisäiset tiedot ovat tietoja, joilla pystytään kuvaamaan kaikki luokan oliot. Tyhjiömittausluokalle niitä on kertynyt viisi. Jokaisella oliolla on siis omat nimet, tietolähteet, yksiköt, näytteenottojen määrät ja ohjaintyypit. Seuraavassa kuvassa on esitetty, miltä näyttää tyhjiömittausluokan sisäisten tietojen klusteri:



Kuva 9. Tyhjiömittausluokan sisäisen datan klusteri: jäsenet datatyypeittäin: *Name* on *string*. *DAQmx Task Name* on lista sisääntuloista. *Samples* on I32. *Unit* ja *Controller Type* ovat enumeraattoreja/listoja.

Tietojen tyypit valittiin seuraavalla periaatteella niistä tiedoista, joilla voi olla vain tietyt vaihtoehdot. Niistä on tehty tyypinmääritysvalikko (*TypeDef.*), *Unit* ja *Controller Type* ovat esimerkkejä siitä. *Unitista* löytyy seuraavat vaihtoehdot: Torr, mbar, Pa, microns ja Volts. Se on lista yksiköistä, joissa oliot palauttavat painearvot. *Controller Type* sisältää seuraavat ohjaintyypit: *Undefined Type*, *HPS Series 937A, IGM-402 ja VGC-301A*. Niiden perusteella ohjelma valitsee tarvittavan laskutoimituksen signaalin käsittelyssä. Tyypinmääritys toimii *case*-valintarakenteen ohjaajana, eli jos kytketään sellainen tyypinmäärityskontrolli *case*-valintarakenteen, se luo automaattisesti toimintaikkunan jokaiselle listatulle vaihtoehdolle.

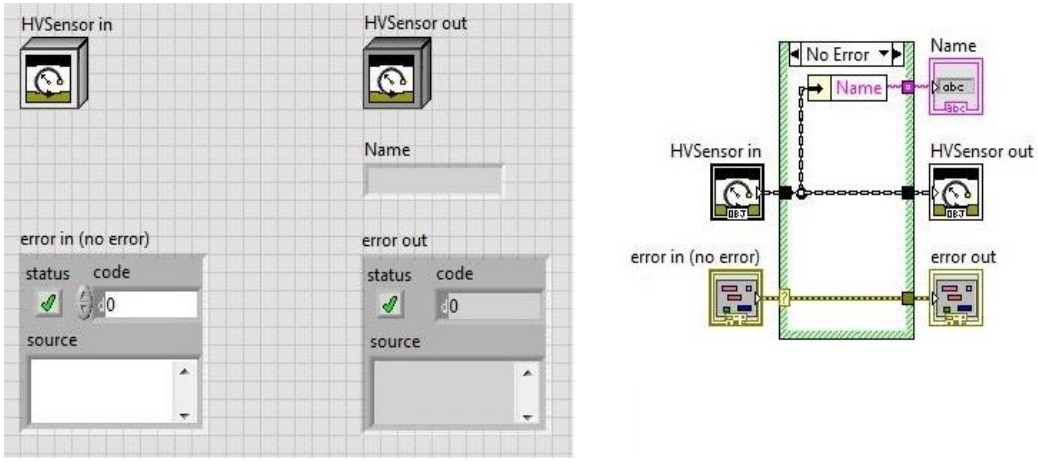
Julkisia muuttujia tyhjiömittausluokassa ei käytetty lainkaan, koska niitä ei tarvittu. Yleensäkin pyritään välttämään niiden käyttöä, sillä ne osoittautuivat epäluotettaviksi käytettynä isohkoissa projekteissa.

4.1.4 Luokan aliohjelmat

Luokan aliohjelmat ovat luokan metodeja, eli kaikkia niitä toimintoja, jotka tapahtuvat luokan sisällä. Ohjelmointityylin mukaan aliohjelmista tehtiin mahdollisimman yksinkertaiset.

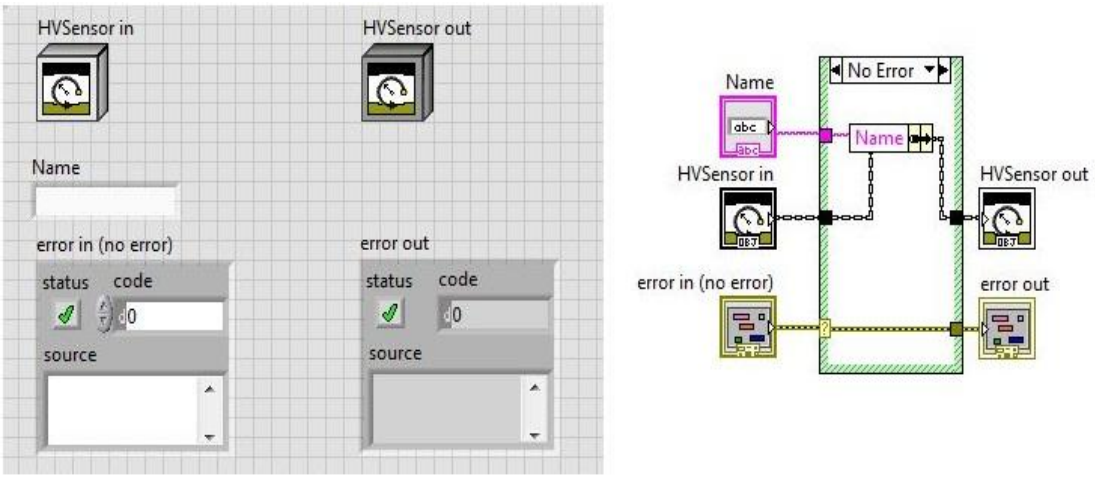
Luku ja kirjoitus metodit/aliohjelmat

Luku- ja kirjoitusaliohjelmat kuvaavat niitä metodeja, jotka ovat tekemisissä luokan datatiedostojen kanssa: ne lukevat niitä tai kirjoittavat niihin. Seuraavissa kuvissa (kuvat 10 ja 11 s. 17 ja 18) esitetään esimerkit molemmista vaihtoehdoista. Näiden aliohjelmien tarkoitus on lukea ja kirjoittaa luokan dataklusterin tiedot.



Kuva 10. Aliohjelman *get Name.vi* etupaneeli ja lohkokaavio; *Unbundle by name* -alifunktiolla toteutettu klusterin purku

Aliohjelma (kuva 10) toteuttaa *Name*-datatiedon lukua. *HVSensor in* ja *HVSensor out* väliin vedetty johto kuvaa luokan dataklusterin, eli kaikki klusterin tiedot ovat siinä johdossa nipussa. Luku ja kirjoitus tapahtuvat nimenomaan tämän johdon purkamisella ja niputtamisella. Tätä varteen LabVIEW'sta löytyy erilaiset purku- (*unbundle*) ja niputus- (*bundle*) -alifunktiot.



Kuva 11. Aliohjelman set *Name.vi* etupaneeli ja lohkokaavio; *Bundle by name* –alifunktiolla on toteutettu klusterin niputus (tietojen korvaus)

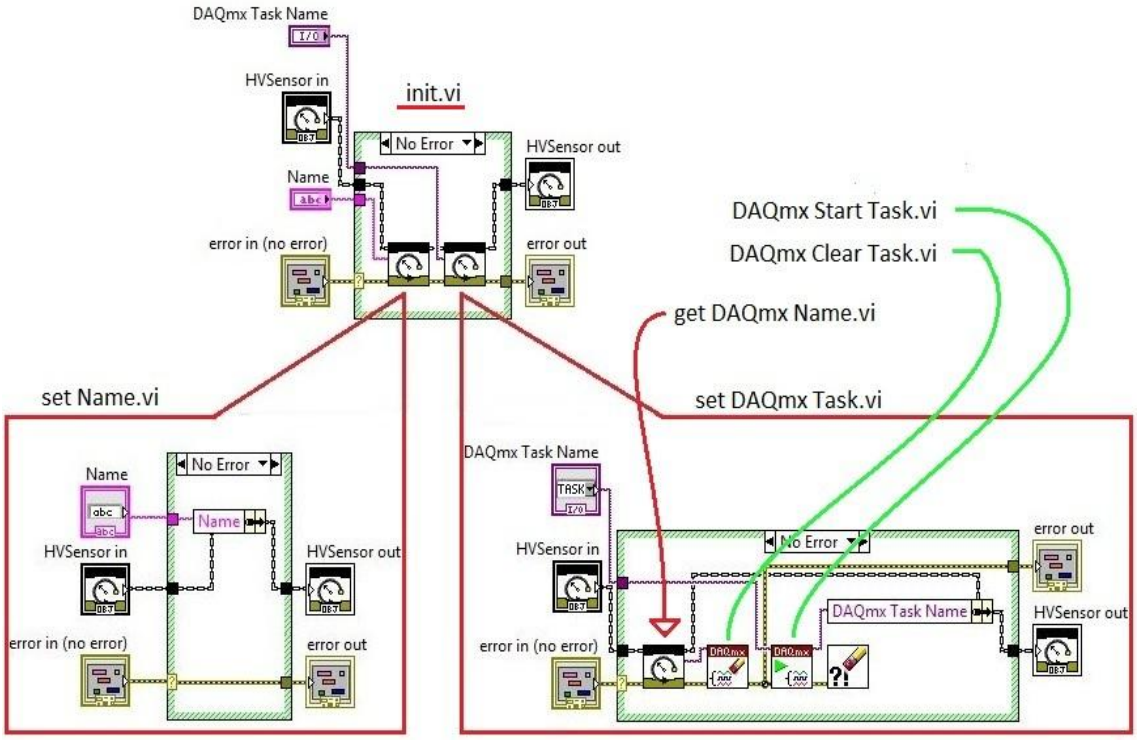
Kuvassa 11 tapahtuu tietojen kirjoitus klusteriin. *Bundle by name* tavallaan korvaa klusterin johdossa *Name*-tiedon uudella kontrollerin nimitiedolla.

Jokaista klusterin tietoa kohti luokkaan on luotu kaksi *get/set*-aliohjelmaa, eli yhteensä kymmenen kappaletta. *get*-etuliitteellä varustettu aliohjelma lukee tiedot ja *set*-liitteellä kirjoittaa. Kaikki *get/set*-aliohjelmat ovat rakenteeltaan samanlaiset.

(Kaikki luokan aliohjelmien ja esimerkkiohjelmien selostustiedostot ja koodit, ks. liite.)

Alustus metodit/aliohjelmat

Kun luodaan uusi olio, tietoklusteria pitää alustaa ja liittyvät tehtävät on käynnistettävä. Ennen olion lopettamista käynnistetyt tehtävät on lopetettava. Nämä ovat *init/kill*-aliohjelmien tehtävä. Tyhjiömittausluokassa ennen olion käynnistymistä *init.vi* alustaa tietoklusterin tiedot oletusarvoihin ja käynnistää tiedonkeruutehtävän.



Kuva 12. Aliohjelman *init.vi toteutus* ja sen kahden aliohjelman rakenne haarauutettuna

Kuva 12 esittää *init.vi*-aliohjelman lohkokaaviota. Kaaviosta voidaan nähdä tietojen alustus ja tehtäväkäynnistys. Nimi (*Name*) alustetaan omassa *set Name.vi* metodiohjelmassa ja tiedonkeruutehtävän nimi (*DAQmx Task Name*) alustetaan tiedonkeruutehtävän käynnistykseen yhteydessä *set DAQmx Task.vi* metodiohjelmassa.

Kun lopetetaan, olio käynnistää *kill.vi*-metodiohjelman, joka sulkee tiedonkeruutehtävän *DAQmx Stop Task.vi* -alifunktiolla.

(Kaikki luokan aliohjelmien ja esimerkkiohjelmien selostustiedostot ja koodit, ks. liite.)

Tietojenkäsittely, metodit/aliohjelmat

Tietojenkäsittelymetodit nimensä mukaan käsittelevät luokan tietoja. Tyhjiömittausluokan tietojenkäsittelymetodit käsittelevät anturiohjaimen signaalia asetuksien mukaan. Tietojenkäsittelymetodeja pystyy lajittelemaan toiminnallisuuden mukaan, mutta tässä projektissa niitä on liian vähän jaettavaksi erillisiin haaroihin.

Tyhjiömittausluokassa on seuraavat tietojenkäsittelymetodit:

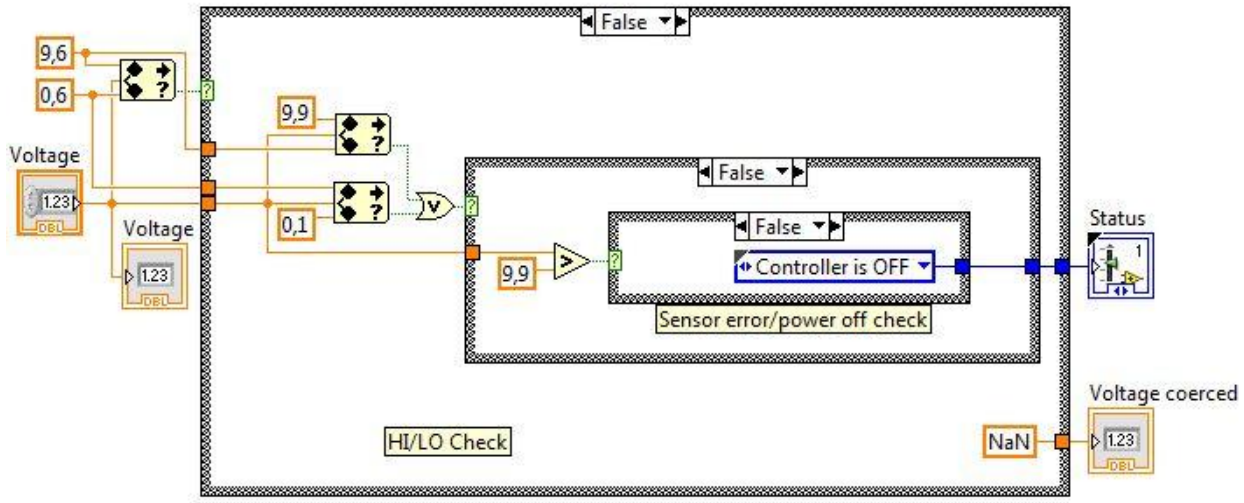
sensor Output status (HPS937A).vi
sensor Output status (IGM-402).vi
sensor Output status (VGS-301A).vi

Kolme edellä mainittua metodia tarkkailevat signaalia ja palauttavat ohjaimen toiminta-alue arvon. Kolmen ohjaimen toiminta-alueet ovat erilaiset, mutta niitä voi kuvailla samoilla tilanimillä. Sitä varten luokan rakenteeseen on luotu luettelo (*enumeraattori*) tiloista, joissa ohjaimet toimivat. Vaihtoehtoja on kuusi (*Undefined status*, *Sensor error*, *HI, LO, Normal Range, Controller is Off*). Ohjaimen HPS937A esimerkin avulla saa kuvan tilamääritysmetodin toiminnasta. Taulukossa 5 esitetään ohjaimen tila-alueet:

Taulukko 5. HPS937A-ohjaimen tila-aluetaulukko (4 s. 20)

If Logarithmic Output Is	*Then*
0.6 to 9.6 V	Normal range
0.2 V	Sensor exposed to a pressure less than in its measurement range (LO displayed)
9.8 V	Sensor exposed to a pressure higher than in its measurement range (HI displayed)
10 V	Thermal conductivity sensor not connected properly or filament is broke
10V	Cold cathode high voltage disabled
10V	No sensor on a channel
10V	Immediately after the Series 937A Controller is switched on
0 V	When the Controller is not powered, but never when power is on

HPS937A-ohjaimella on viisi tilaa. Normaalialue on 0.6 - 9.6 V, jolloin tilaselvitys aliohjelma palauttaa *Normal Range* tilan. Tila-alueet 0.1 - 0.6 V ja 9.6 - 9.9 V vastaavat *LO-* ja *HI* -tiloja. Alle 0.1 V:n signaali vastaa *Controller is Off* -tilaa. Yli 9.9 V:n signaali viittaa siihen, että ohjaimessa tai anturissa on jokin vika, silloin ohjelma palauttaa *Sensor Error* tilan. Kuvassa 13 esitetään, miten kyseisen ohjaimen tilaseuranta on toteutettu G-kielellä.



Kuva 13. HPS937A-ohjaimen *Controller is Off* -tilaselvityksen koodi

Tilaselvitysohjelman kuvassa (kuva 13) on esitetty HPS937A-ohjaimen *Controller is Off* (ohjain on pois päältä) -tilakartoitus. Ohjelman algoritmi on seuraava: ensin ohjelma tarkistaa, onko signaali 0.6 - 9.6 V alueen sisällä. Kun vastaus on negatiivinen, ohjelma tarkistaa, onko signaali 0.1 - 0.6 V ja 9.6 - 9.9 V. Tämän jälkeen ohjelma tarkistaa, onko se suurempi kuin 9.9 V, jos ei ole, se tarkoittaa, että signaali on alle 0.1 V, mikä vuorostaan tarkoittaa, että tila vastaa 0 V:a, eli ohjain on pois päältä.

Seuraavat tietojenkäsittelymetodit:

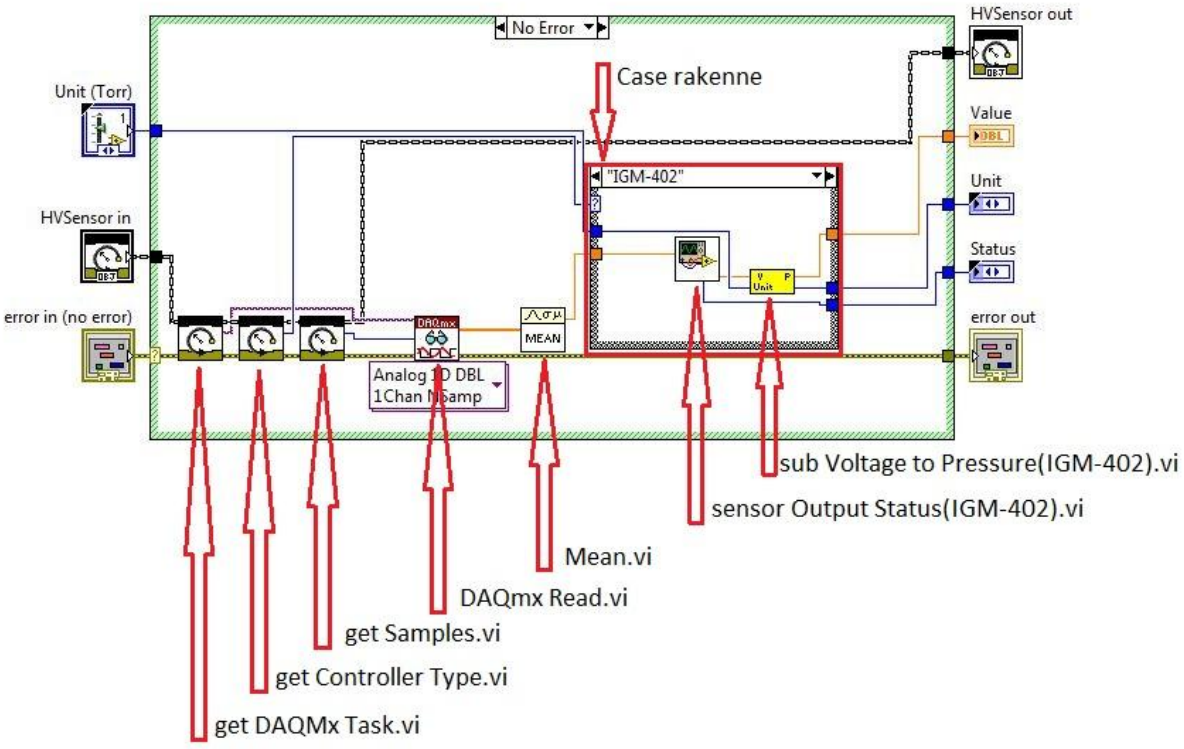
sub Voltage to Pressure (IGM-402).vi
sub Voltage to Pressure (HPS937A).vi
sub Voltage to Pressure (VGS-301A).vi

Aliohjelmat/metodit *sub Voltage to Pressure(*).vi* käsittelevät ohjaimien jännite-paine muunnosta. G-kielen yhtälöt ja toteutukset ovat tämän työn *Ohjaimet*-osiosssa (ks.4.1.2; kuvat 4, 5, 6. s. 11, 12, 13).

Seuraava tietojenkäsittelymetodi:

read Pressure.vi

Aliohjelma *read Pressure.vi* nimensä mukaisesti lukee painearvon. Ohjelma lukee sisääntuloa ja palauttaa painearvon riippuen sisääntulovalinnasta, ohjaimesta ja yksikkövalinnasta.



Kuva 14. Aliohjelman *read Pressure.vi* rakenne; välivaiheet ovat merkitty nuolilla

Aliohjelma *read Pressure.vi* :n ja sen aliohjelmien toimintaperiaate on seuraava:

- Jos kyseessä ei ole virhetila, ensimmäinen metodi *get DAQMx Task.vi* hakee instanssiluettelosta datalähteen ja välittää sen *DAQmx Read.vi* aliohjelmalle
- Seuraava *get Controller Type.vi* hakee instanssiluettelosta ohjaintyyppiin ja välittää sen *Case*-rakenteelle.
- Seuraavaksi *get Samples.vi* hakee instanssiluettelosta näytteenottojen määrän ja välittää sen *DAQmx Read.vi* -aliohjelmalle.

- *DAQmx Read.vi* vuorollaan, kun se on saanut kaikki tarvittavat tiedot, palauttaa valitun tietolähteen signaalin ja välittää sen *Case*-rakenteelle keskiarvofunktion *Mean.vi* kautta
- Riippuen ohjaimen tyypistä *Case*-rakenne valitsee sitä vastaavan tilan (tässä tapauksessa IGM-402 -ohjaimen) ja suorittaa sen sisältöä eli selvittää ohjaimen tilan *sensor Output Status(IGM-402).vi* avulla sekä suorittaa jännite-paineeseen muunnoksen *sub Voltage to Pressure(IGM-402).vi* avulla.
- Lopuksi ohjelma kokonaisuudessaan palauttaa kolme arvoa: painearvon, yksikön ja ohjaimen tila-arvon.

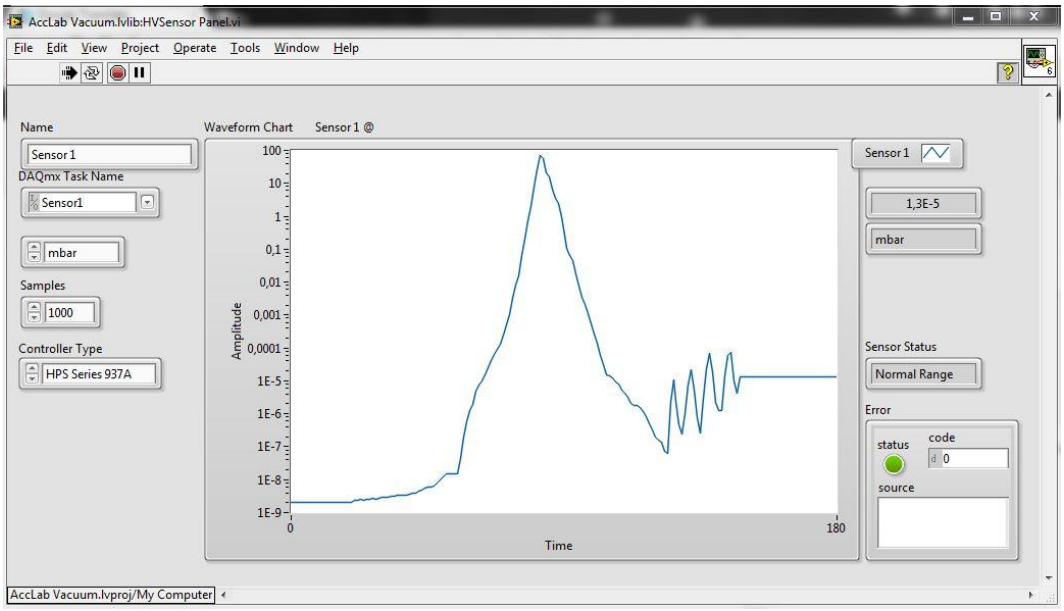
Seuraavat tietojenkäsittelymetodit:

start DAQMx Task.vi
stop DAQMx Task.vi

Start- ja *Stop-DAQMx Task.vi* -metodit käsittelevät tiedonkeruutehtävän käynnistystä ja lopettamista. Tiedonkeruutehtävän käynnistys ja lopetus vaativat jonkin verran koneellista aikaa. Koska tietojen luku tapahtuu silmukassa, tiedonkeruutehtävä on käynnistettävä, ennen kuin ohjelma siirtyy silmukkaan välttääkseen ohjelman ruuhkautumisen. Tiedonkeruutehtävä käynnistetään ennen silmukkaa. Silmukassa sitä vain luetaan ja silmukan jälkeen lopetetaan. Tällä tavalla tehtävä käynnistetään ja lopetetaan vain yhden kerran ohjelman ajon aikana, mikä keventää ohjelmaa merkittävästi.

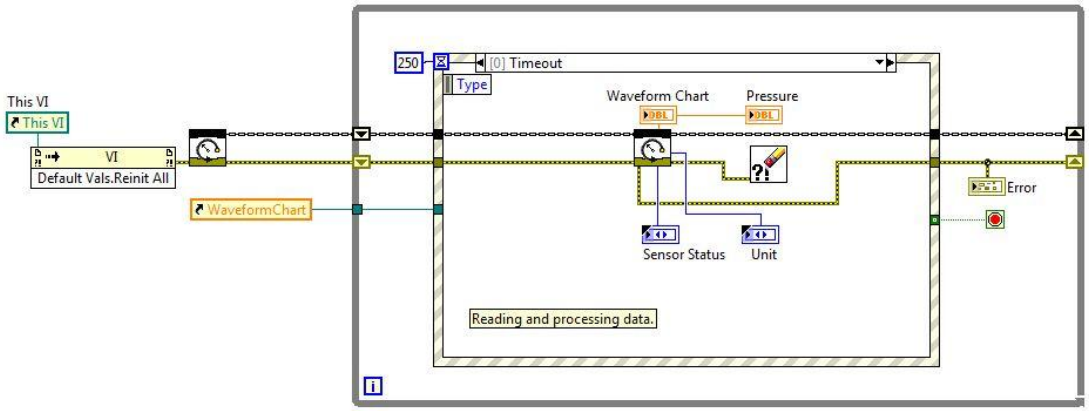
Esimerkkikoodi

Esimerkkikoodiosiossa (*Example code*) esitetään esimerkkiohjelmat, jotka antavat kuvan siitä, miten luokkaa käytetään. Tämän luokan esimerkkiohjelmista on kehitetty käyttöliittymäohjelma, joka kuvaa yhden anturin toimintaa ja palauttaa painearvon lukuna ja piirtää siitä käyrän käyttäjän antamien asetuksien mukaan.

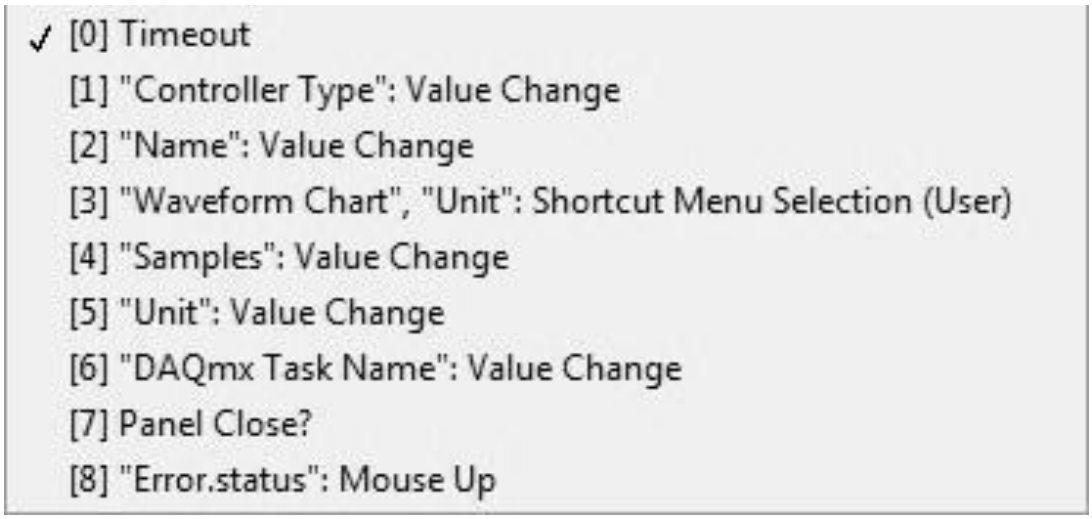


Kuva 15. Tyhjiömittausluokan esimerkkiohjelman *HVSensor Panel.vi* käyttöliittymä

Kuvan 15 ohjelma on hyvä esimerkki siitä, miten tämän luokan olioita käytetään tarkoituksenmukaisesti. Ohjelma esittää olion perustoiminnot. Kuvasta näkee, että kaikki olion datakontrollerit on sijoitettu paneelin vasemmalle laidalle (nimi, tiedonkeruutehtävän nimi, yksikkö, näytteenottojen määrä, ohjain tyyppi), oikealle olion palautteet (paine lukuna, yksikkö, ohjaimen tila, virheikkuna) ja keskellä aaltomuotokaavio, joka piirtää painekäyrän suhteessa aikaan.



Kuva 16. Tyhjiömittausluokan esimerkkiohjelman *HVSensor Panel.vi* -lohkokaavio



Kuva 17. Tyhjiömittausluokan esimerkkiohjelman *HVSensor Panel.vi* -lohkokaavion sisältämä *case*-rakenteen tilaluettelo

Kuvissa 16 ja 17 (s. 24 ja 25) on esitetty käyttöliittymäesimerkkiohjelman koodi ja *case*-rakenteen tilaluettelo. Kun ohjelma on käynnissä ja mitään ei tapahdu käyttöliittymän ikkunassa, ohjelma on perustilassa lukemassa ja käsittelemässä tietoja. Jos jokin tilaluettelon tiloista toteutuu, ohjelma käy suorittamassa kyseistä tilaa vastaavat toiminnot ja palaa perustilaan. Esimerkiksi, kun käyttäjä päättää vaihtaa anturin nimen (*Name*), hän vaihtaa sen käyttöliittymässä, jolloin ohjelma huomaa tämän tapahtuman ja käy kirjoittamassa datatiedostoon (*Name*) uuden nimen ja päivittää käyttöliittymän sen mukaisesti. *case*-valintarakenteen tilaluetteloon on listattu kaikki tarvittavat tapahtumavaihtoehdot. Tuleva luokan käyttäjä voi muokata valmista koodia tarkoituksensa mukaisesti tai kirjoittaa oman koodinsa käyttäen esimerkkikoodin osion (kuva 8, s. 15) aliohjelmia.

4.1.5 Olion rajapinta

Rajapintakäsite ohjelmoinnissa tarkoittaa työkaluja, toimintamenetelmiä, yhteisvaikutuksen sääntöjä ja hallinta sekä valvonta -algoritmeja järjestelmän jäsenten välillä. Kaikki vuorovaikutukset olion sisätietojen kanssa tapahtuu metodien kautta, eli jos on tarve muokata tietoja datatiedostossa (*HVSensor.ctr*), siihen käytetään kirjoitus/luku (*get/set*) metodeja. Tästä seuraa, että olion rajapinta on kyseisen luokan metodit.

5 Dokumentointi

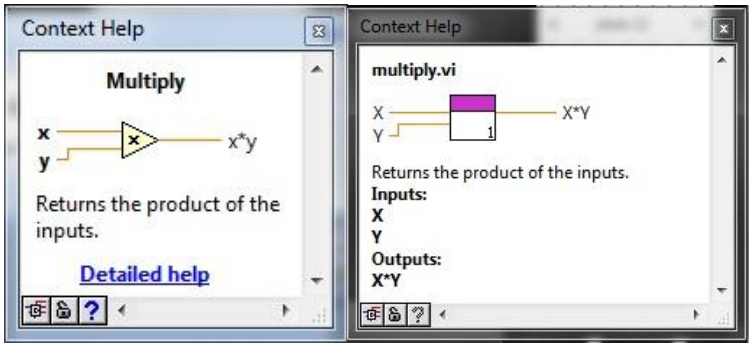
Ohjelmiston dokumentointi on ohjelman mukana tulevan asiakirjan laatiminen, jossa kuvataan ohjelman toiminnallisuutta, ja miten sitä kuuluu käyttää. Dokumentoinnin voi jakaa seuraaviin tyyppeihin:

- *arkkitehtuurinen/projekti-dokumentaatio*; kuvaa ohjelmaa/projektia pääpiirteittäin, työympäristöä ja toteutusperiaatteita; vastaa kysymykseen *Miksi juuri näin?*
- *tekninen dokumentointi*; selvittää ohjelmakoodin, algoritmit ja rajapinnat; vastaa kysymyksen *Miten on toteutettu?*
- *käyttäjädokumentaatio* on loppukäyttäjälle tarkoitettu manuaali, joka kertoo enemmän ohjelman käyttämisestä kuin sisäisestä rakenteesta
- *markkinointidokumentaatio* on kaupallisille tuotteille tarkoitettu eli mainostaa tuotetta (7 s. 3).

Koska tyhjiömittausluokka on toisen projektin osa, sitä ei dokumentoitu kaikkien dokumentointityyppien mukaisesti, vaan keskityttiin pääosin tekniseen dokumentointiin. Arkkitehtuurinen dokumentointi jää tarpeettomaksi, sillä työympäristö ja toteutusperiaatteet on määritelty ennalta isäntäprojektissa. Käyttäjälle tarkoitetulla dokumentaatiolle ei ole tarvetta dokumentaatioon, koska se on luokka eikä ohjelma, eikä tässä projektissa ole varsinaista käyttöliittymää. Lisäksi luokka on tarkoitus siirtää kokeneelle ohjelmoijalle, joka osaa erottaa tarvittavat tiedot teknisesti dokumentoiduista esimerkkialiohjelmista. Markkinoinnista ei puhuta, sillä luokka on osa projektia, joka ei ole kaupallinen.

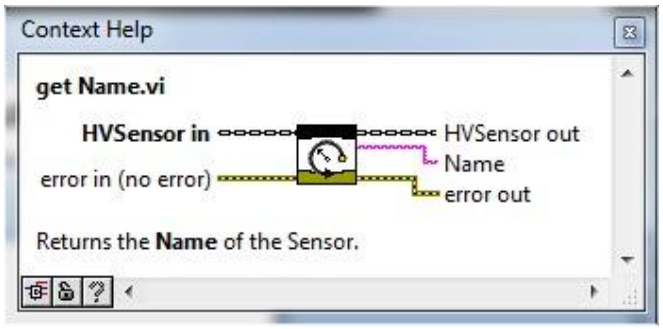
5.1 Taustaselostus (*Context Help*)

LabVIEW'n ohjelmassa on taustaselostusoptio, joka esittää ja selittää kaikki koodin komponentit. Kun ohjelmoija osoittaa jotain alifunktiota tai aliohjelmaa hiiren osoittimella, taustaselostus näyttää erillisessä ikkunassa sen dokumentointitiedot. Kaikista alifunktioista ja työkaluista, jotka löytyvät valmiiksi ohjelmointiympäristöstä, on laadittu taustaselostus valmiiksi. Ohjelmoijan kehittämät aliohjelmat on dokumentoitava. Kuva 18 esittää valmiin alifunktion ja toiminnaltaan samanlaisen aliohjelman selostusikkunat:



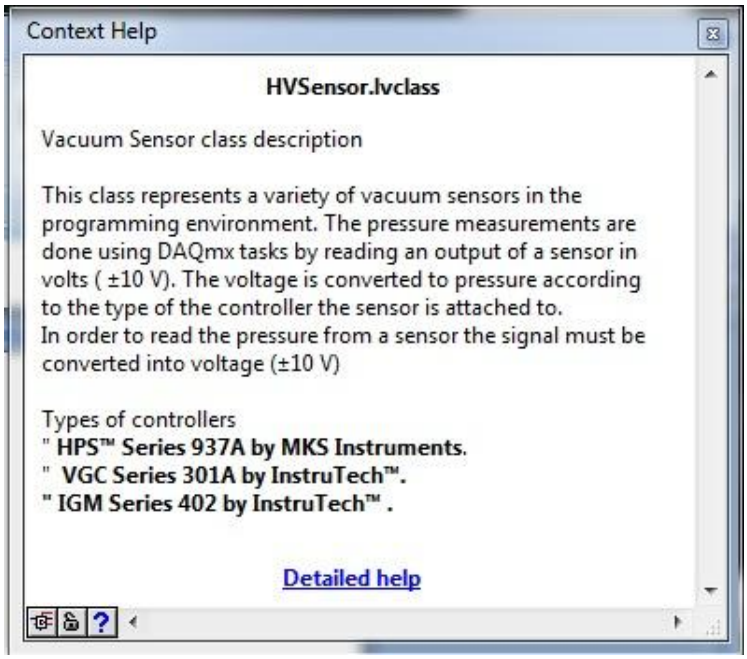
Kuva 18. Kertolaskualifunktion (*multiply* vasemmalla) ja kertolaskualiohjelman (*multiply.vi*) selostusikkunat

Hyvä tekninen dokumentointi edellyttää, että kaikkiin luokan aliohjelmiin laaditaan kuvaava selostus. Jokaiselle aliohjelmalle kannattaa laatia myös kuvaava tiedoston nimi, joka helpottaa niiden järjestelyä luokan kirjastossa ja antaa ohjelmoijalle kuvan niiden toiminnallisuudesta jo pelkän nimen perusteella. Esimerkkinä siitä voidaan pitää *get/set*-metodeja. Aliohjelmien *get Name.vi* ja *set Name.vi* etuliitteet *get* (saada) ja *set* (asettaa) viittaavat siihen, että ohjelma hakee nimessä mainittuun (*Name*) tiedon dataklusterista tai kirjoittaa siitä siihen.



Kuva 19. *get Name.vi* aliohjelman selostusikkuna

Kuvassa 19 (s. 27) on esitetty *get Name.vi* -aliohjelman selostusikkuna, jossa on metodin nimi, piirrosmerkki, rajapinta ja selostus: *Returns the **Name** of the Sensor* (palauttaa anturin nimitiedon). Samalla tavalla dokumentoidaan kaikki luokan metodit ja datatiedostot, sekä luokan sisällä, että sen ulkopuolella, mutta samassa kirjastossa olevat. Laajemmissa ohjelmissa selostukseen ei ole mahdollista laittaa kaikki tarvitsemat tiedot, minkä takia sinne laitetaan ainoastaan tärkeimmät tiedot ja loput sijoitetaan erillisen *help*-tiedostoon, johon osoittava oikotielinkki esiintyy selostusikkunan *Detailed help* –linkinä (kuva 20).

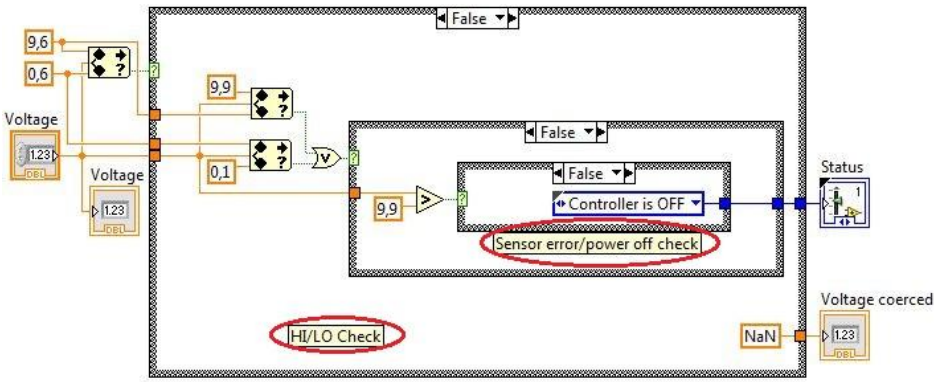


Kuva 20. *HVSensor.lvclass* tyhjiömittausluokan selostusikkuna

(Kaikki luokan aliohjelmien ja esimerkkiohjelmien selostustiedostot ja koodi, ks. liite.)

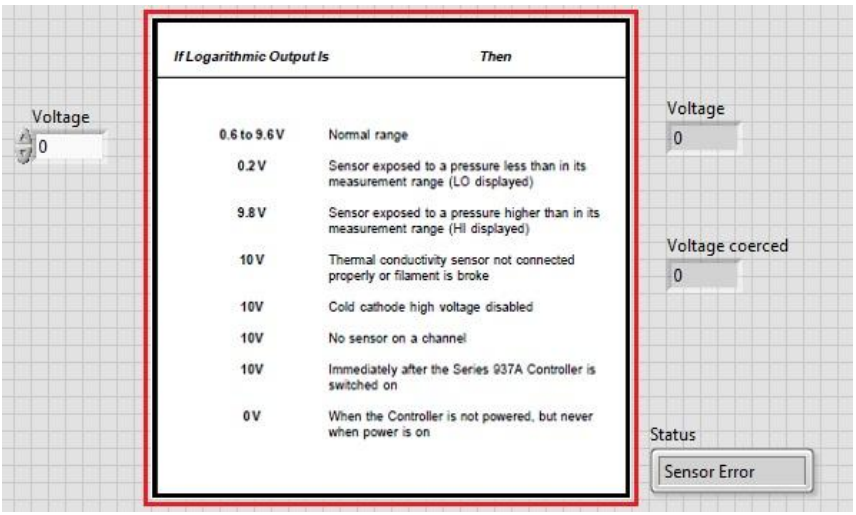
5.2 Lohkokaaviodokumentointi

Isoissa ohjelmissa, joissa on paljon koodia samassa lohkokaaviossa, selkeys on olennaista. Samoin, kuin tekstipohjaisissa kielissä, LabVIEW'ssa on mahdollisuus selostaa koodia suoraan sen viereen. Hiiren kaksoispainalluksella lohkokaaviotaustalle voidaan kirjoittaa siellä tarvittavat opastusvirkkeet. Etupaneeli-ikkunaan voidaan tämän lisäksi liittää kuvia. Molemmat vaihtoehdot esitetään kuvissa 21 ja 22 (s. 29).



Kuva 21. Ohjaimen HPS937A *Controller is Off* tilaselvityksen koodi; punaisella on merkitty lohkokaavioselostukset

Kuvassa 21 ohjelmakoodiin on merkitty ohjelman tilaselostukset *HI/LO Check* ja *Sensor error/power off check*, jotka viittaavat *Case*-valintarakenteiden toimintoihin. Ensimmäinen (*HI/LO Check)* tarkistaa, onko anturi *HI* tai *LO* -tilassa, toinen (*Sensor error/power off check)* tarkistaa, onko siinä virhe, tai onko se pois päältä. Tällä selostustavalla mahdollistetaan ohjelman nopea ymmärtäminen.



Kuva 22. *sensor Output Status(HPS 937A).vi* aliohjelman etupaneelinäkymä, johon on liitetty punaisella merkitty tilataulukko

(Kaikki luokan aliohjelmien ja esimerkkiohjelmien selostustiedostot ja koodit, ks. liite.)

6 Yhteenveto

Työssä oli tavoitteena kehittää tyhjiömittausmoduuli suurta isäntäohjelmaa (Helsingin yliopiston kiihdytinlaboratorion käyttöjärjestelmä) varten olio-ohjelmointia hyödyntäen LabVIEW-ohjelmointiympäristössä G-ohjelmointikielellä. Työtä varten tutkittiin graafisen ohjelmointikielen ominaisuuksia ja luokkapohjaisen olio-ohjelmoinnin periaatteita.

Edellytyksenä työn onnistumiseen oli tavoitteen tarkka ymmärtäminen. Aiheluokan kuului olla selkeä ja järjestelmällinen. Toiminta rajoittui tarkasti tyhjiömittaukseen. Pääajatuksena oli helppo ja varma integroitavuus.

Graafinen ohjelmointikieli ja sen luokkapohjainen olio-ohjelmointi osoittautuivat hyvin vahvoiksi ohjelmointityökaluiksi. Ohjelmoiminen on nopeaa ja järjestelmällistä. Luokkapohjainen ohjelman rakenne on hyvin selkeä ja muokkausystävällinen. Se soveltuu erityisesti isojen ohjelmien rakentamiseen.

Työn tulokseksi saatiin tyhjiömittausluokka, joka kattaa kaikki sille työn alkuvaiheessa asetetut vaatimukset. Luokka kehittyi kolmessa vaiheessa: ensimmäisessä kartoitettiin luokan sisäiset tiedot ja lisättiin luokan dataklusteriin; toisessa kehitettiin sisäisiä tietoja käsitteleviä aliohjelmia (metodeja) ja kolmannessa kehitettiin signaalinkäsittelyaliohjelmia. Luokalle tehtiin myös oheis/esimerkki-ohjelmaosio, joka sisältää luokan oliota käyttävä ohjelma ja sen oheisohjelmat. Esimerkkiohjelman tehtävänä on antaa luokan käyttäjälle käsityksen siitä, miten luokkaa ja sen metodeja käytetään.

Koska luokka on tarkoitettu seuraaville kehittäjille tulevaisuudessa, luokka dokumentoitiin hyvin huolellisesti. Koska luokka on osa isoa projektia, sille on laadittu ainoastaan tekninen dokumentointi.

Luokka integroidaan kiihdyttimen käyttöjärjestelmään sen käyttöönoton yhteydessä, mutta se on valmis toimintaan sellaisenaankin. Esimerkkiohjelma pystyy käsittelemään mittaustuloksia erillisenäkin ohjelmana. Kehitysmahdollisuutena voitaisiin ajatella sellaista tilannetta, jossa oliot olisivat erillisillä tietokoneilla tyhjiöanturien välittömässä läheisyydessä ja näin lähettäisivät mittaustulokset palvelimelle sisäisen verkon kautta.

Lähteet

-1. Larsen, Roland W**.** *LabVIEW for Engineers.* s.l. : Published by Prentice Hall, 2010. isbn-13:978-0-13-609429-6.

-2. Kerry, Elijah, An Introduction to LabVIEW Object-Oriented Design Patterns. https://decibel.ni.com/content/docs/DOC-15014. *https://decibel.ni.com.* [Online] 1. 09 2011. [Viitattu: 12. 12 2011.] https://decibel.ni.com/content/servlet/JiveServlet/download/15014-18-31900/LabVIEW%20Object%20Oriented%20Design%20Patterns.pptx.

-3. Rantanen, Sari Saxholm ja Markku. *Paineen Mittaus.* Espoo : Mittatekniikan keskus, 2011. ISBN 978-952-5610-67-3.

-4. MKS Instruments, Inc., Vacuum Product Group. *HPS Series 937 A High Vacuum Multi-Sensor System, User Manual .* Boulder USA : MKS Instruments, Inc., Vacuum Product Group, 1998. 100009273 Rev. A1.

-5. InstruTech, Inc. *Miniature Ionization Vacuum Gauge With dual convection IGM-402 Module, The "Hornet".* Longmount : InstruTech, Inc., 2009.

-6. InstruTech, Inc. A. *VGC301A Convection Vacuum Gauge Controller, User Manual.* Longmount : InstruTech, Inc., 2011.

-7. F., State Standart of the Rus. *Information technology. Guidelines for the management of software documentation.* Moscow : Standard publisher of Rus.F., 1994. iso 9127-88.

Liite 1

Tyhjiömittausluokan lähdekoodi ja dokumentaatio

Liitteen sisällys

1 Data-aliohjelmat 3

1.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**HVSensor.ctl** 3
1.2 AccLab Vacuum.lvlib:HVSensor.lvclass:Controller **Type Enum.ctl** 3
1.3 AccLab Vacuum.lvlib:HVSensor.lvclass:Pressure **Unit Enum.ctl** 4
1.4 AccLab Vacuum.lvlib:HVSensor.lvclass:Sensor **Status Enum.ctl** 4

2 get/set-aliohjelmat 5

2.1 AccLab Vacuum.lvlib:HVSensor.lvclass:get **Controller Type.vi** 5
2.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**get DAQMx Task.vi** 7
2.3 AccLab Vacuum.lvlib:HVSensor.lvclass:**get Name.vi** 9
2.4 AccLab Vacuum.lvlib:HVSensor.lvclass:**get Samples.vi** 10
2.5 AccLab Vacuum.lvlib:HVSensor.lvclass:**get Unit.vi** 11
2.6 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Controller Type.vi** 12
2.7 AccLab Vacuum.lvlib:HVSensor.lvclass:**set DAQMx Task.vi** 13
2.8 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Name.vi** 14
2.9 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Samples.vi** 15
2.10 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Unit.vi** 16

3 Init/kill-aliohjelmat 17

3.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**init.vi** 17
3.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**kill.vi** 18

4 Sensor out status –aliohjelmat 19

4.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**sensor Output Status (HPS 937A).vi** 19
4.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**sensor Output Status (VGC-301A).vi** 21

4.3 AccLab Vacuum.lvlib:HVSensor.lvclass:**sensor Output Status (IGM-402).vi** 22

5 sub voltage to pressure –aliohjelmat 24

5.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**sub Voltage to Pressure (HPS 937A).vi** 24
5.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**sub Voltage to Pressure (IGM-402).vi** 25
5.3 AccLab Vacuum.lvlib:HVSensor.lvclass:**sub Voltage to Pressure (VGC-301A).vi** 27

6 AccLab Vacuum.lvlib:HVSensor.lvclass:read Pressure.vi 28

7 Start/Stop DAQMx task –aliohjelmat 31

7.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**start DAQMx Task.vi** 31
7.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**stop DAQMx Task.vi** 32

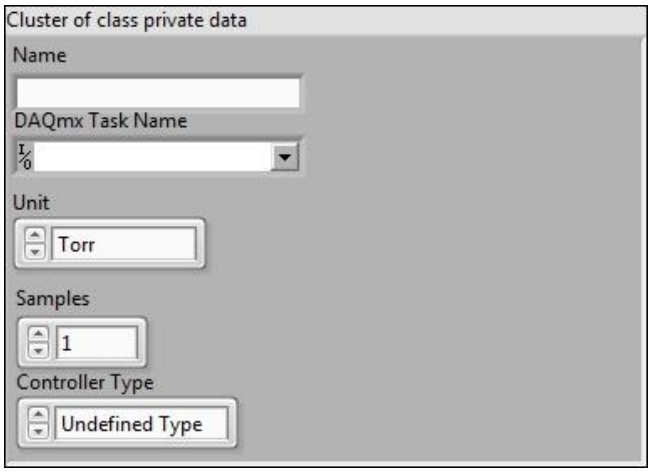
8 Example code –aliohjelmat 33

8.1 AccLab Vacuum.lvlib:**HVSensor Panel.vi** 33
8.2 AccLab Vacuum.lvlib:**handle DAQMx Task Value Change.vi** 37
8.3 AccLab Vacuum.lvlib:**handle Timeout.vi** 38
8.4 AccLab Vacuum.lvlib:**handle Waveform Chart mapping .vi** 39
8.5 AccLab Vacuum.lvlib:**handle Waveform Chart Shortcut Menu Selection (User).vi** 40
8.6 AccLab Vacuum.lvlib:**update Plot Caption.vi** 41
8.7 AccLab Vacuum.lvlib:**update Waveform Chart Caption.vi** 43
8.8 AccLab Vacuum.lvlib:**update Waveform Chart mapping.vi** 44
8.9 AccLab Vacuum.lvlib:**handle Sensor Name Value Change.vi** 45

1 Data-aliohjelmat

1.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**HVSensor.ctl**

Vacuum Sensor object. Private Data Controls.



1.2 AccLab Vacuum.lvlib:HVSensor.lvclass:Controller **Type Enum.ctl**

Controller type Type Definion.

Reserved for future use.

Contains:

Undefined type

HPS Series 937A

VGC-301A

IGM-402

 **Controller Type**

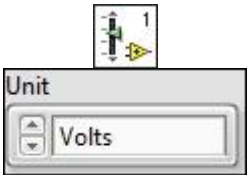
1.3 AccLab Vacuum.lvlib:HVSensor.lvclass:Pressure **Unit Enum.ctl**

Pressure Unit Type Definition.

Reserved for future use.

Contains:

Volts
microns
Torr
Pa
mbar



Unit

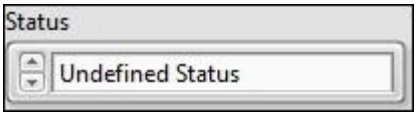
1.4 AccLab Vacuum.lvlib:HVSensor.lvclass:Sensor **Status Enum.ctl**

Sensor Status Type Definition.

Reserved for future use.

Contains:

Undefined Status
Controller is OFF
LO
Normal Range
HI
Sensor Error

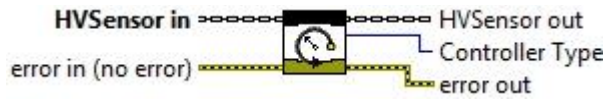


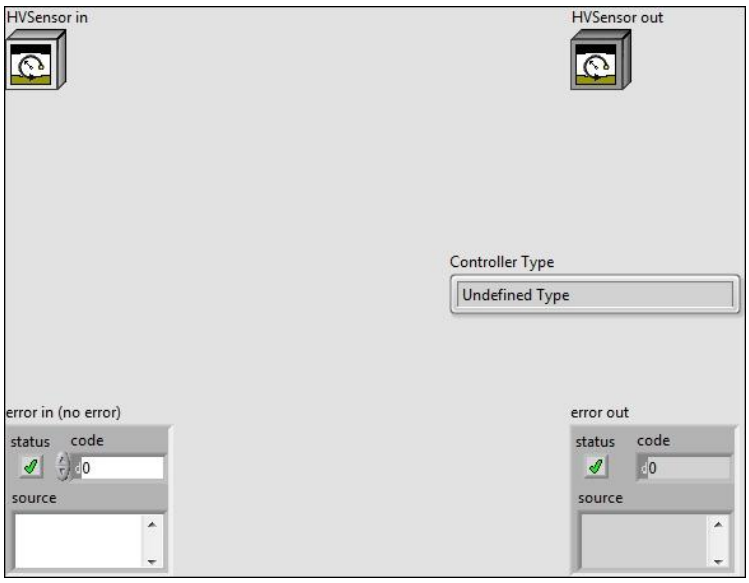
Status

2 get/set-aliohjelmat

2.1 AccLab Vacuum.lvlib:HVSensor.lvclass:get **Controller Type.vi**

Returns the **Controller Type**.





error in (no error)

The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

code

The **code** input identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

source

The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

HVSensor in

error out

The **error out** cluster passes error or warning information out of a VI to be used by other VIs.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

status

The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

code

The **code** input identifies the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

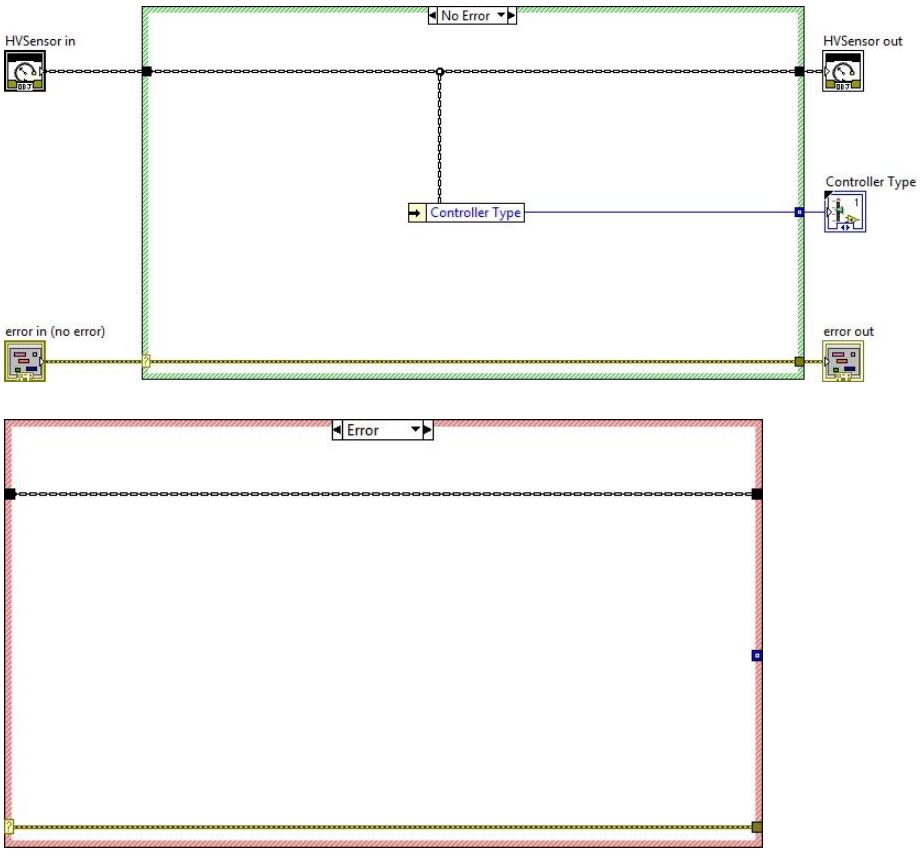
source

The **source** string describes the origin of the error or warning.

The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

HVSensor out

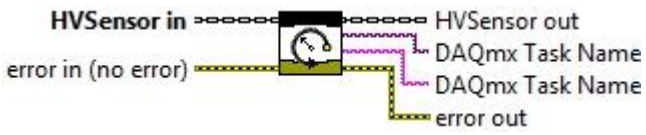
Controller Type

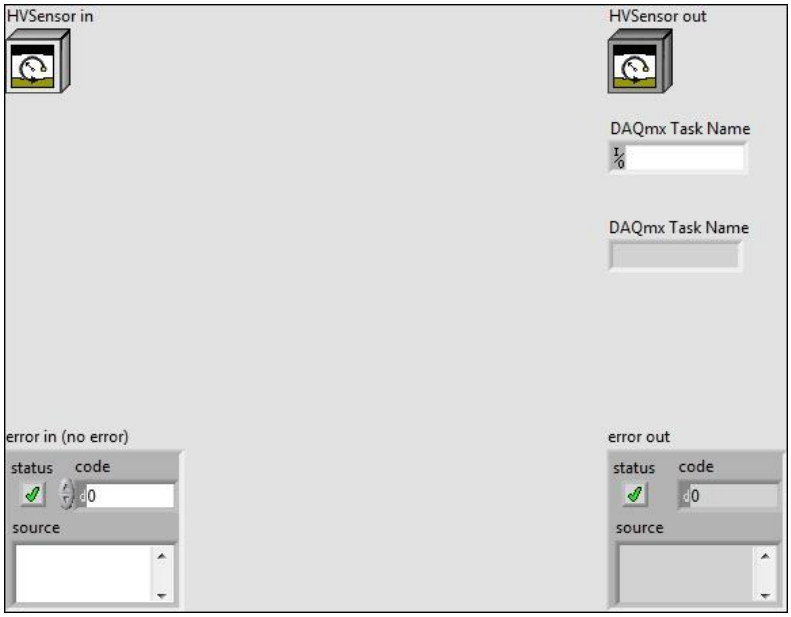


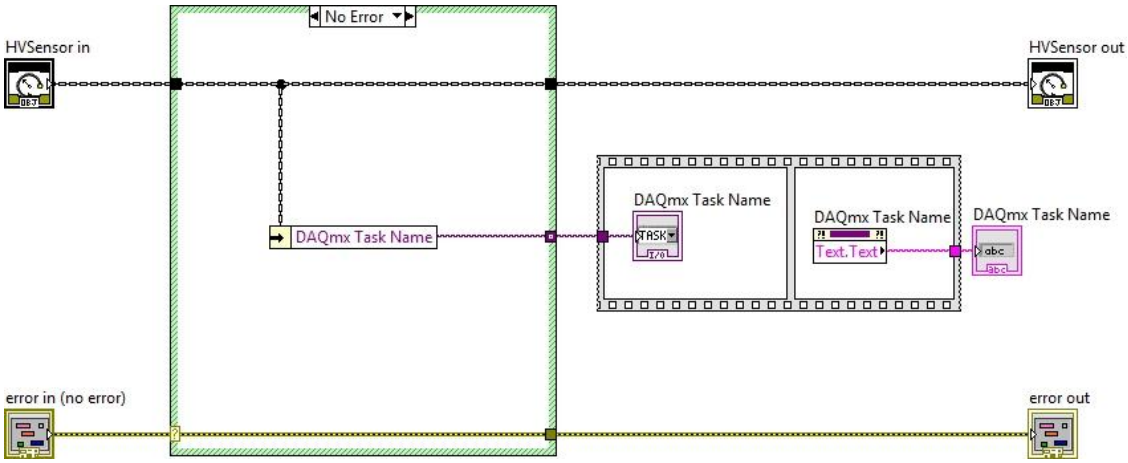
"AccLab Vacuum.lvlib:HVSensor.lvclass:get Controller Type.vi History"

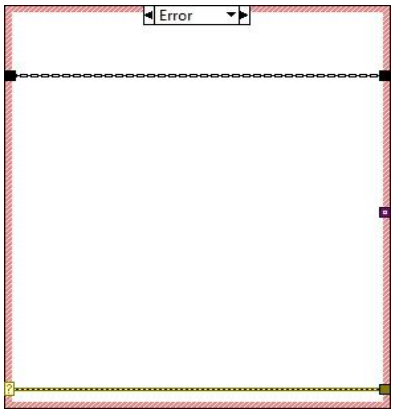
2.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**get DAQMx Task.vi**

Returns **the DAQMx Task Name**.



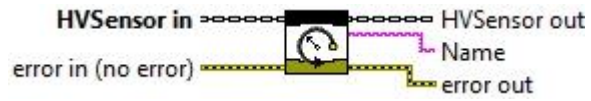


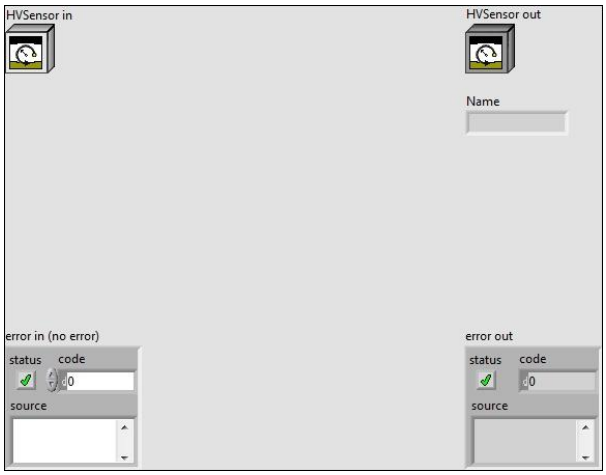


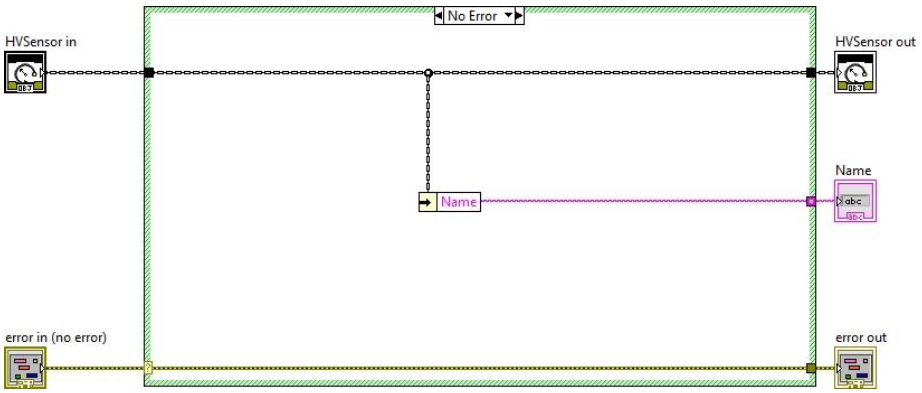


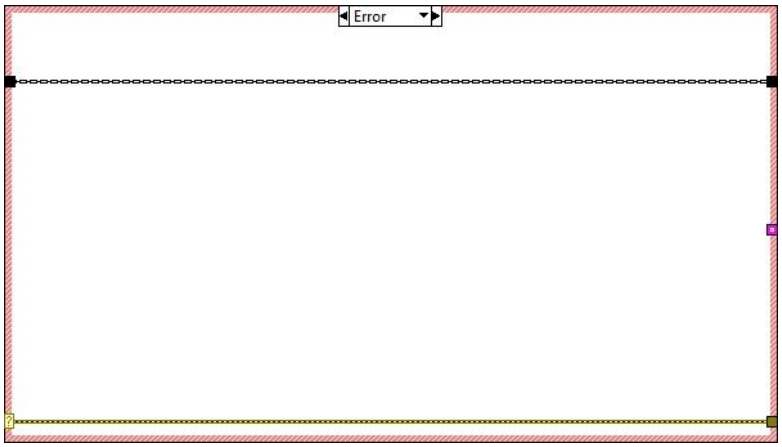
AccLab Vacuum.lvlib:HVSensor.lvclass:get DAQMx Task.vi History"

2.3 AccLab Vacuum.lvlib:HVSensor.lvclass:**get Name.vi**



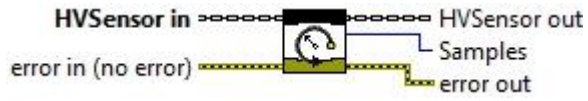


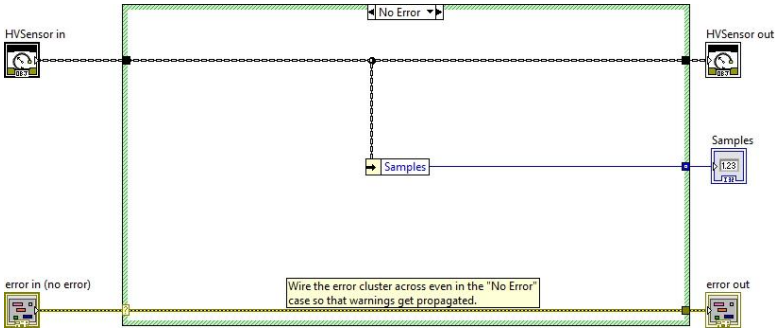


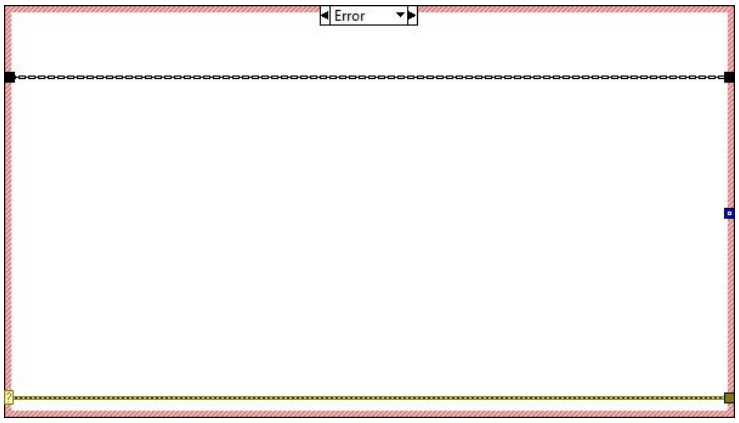


"AccLab Vacuum.lvlib:HVSensor.lvclass:get Name.vi History"

2.4 AccLab Vacuum.lvlib:HVSensor.lvclass:**get Samples.vi**

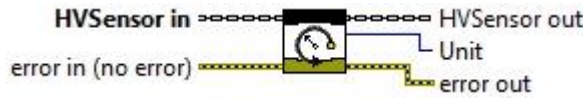


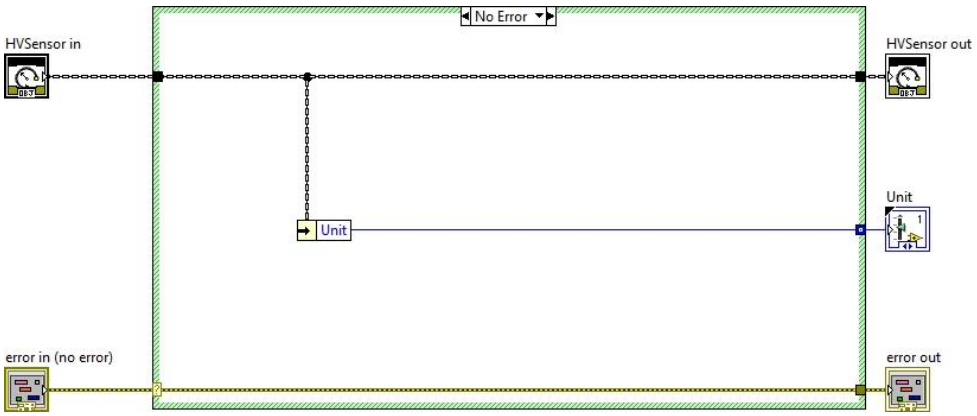


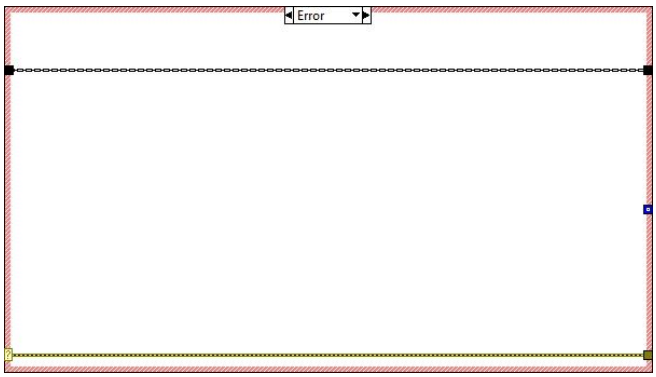


"AccLab Vacuum.lvlib:HVSensor.lvclass:get Samples.vi History"

2.5 AccLab Vacuum.lvlib:HVSensor.lvclass:**get Unit.vi**

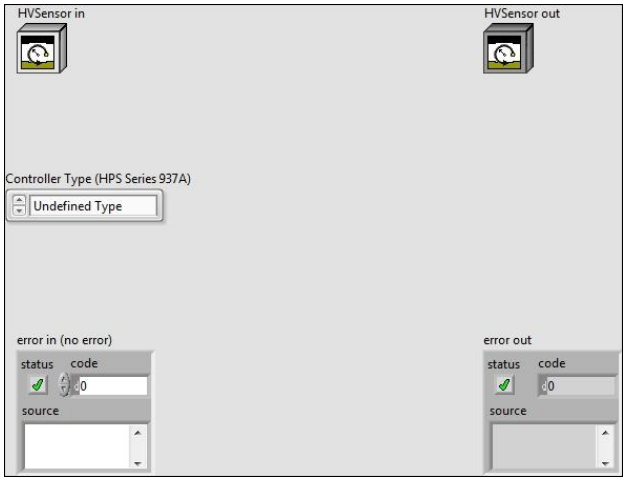


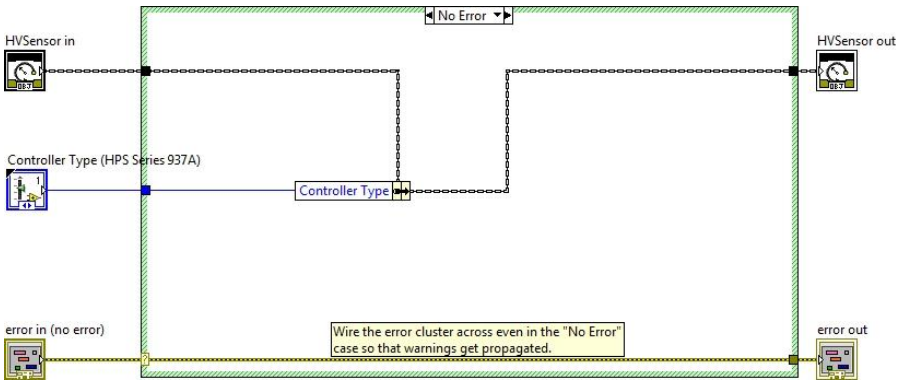


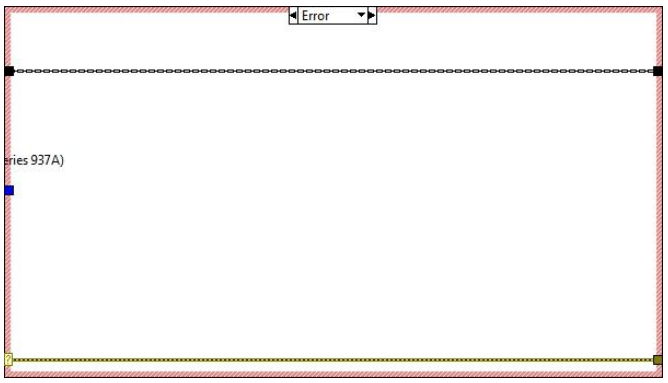


"AccLab Vacuum.lvlib:HVSensor.lvclass:get Unit.vi History"

2.6 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Controller Type.vi**

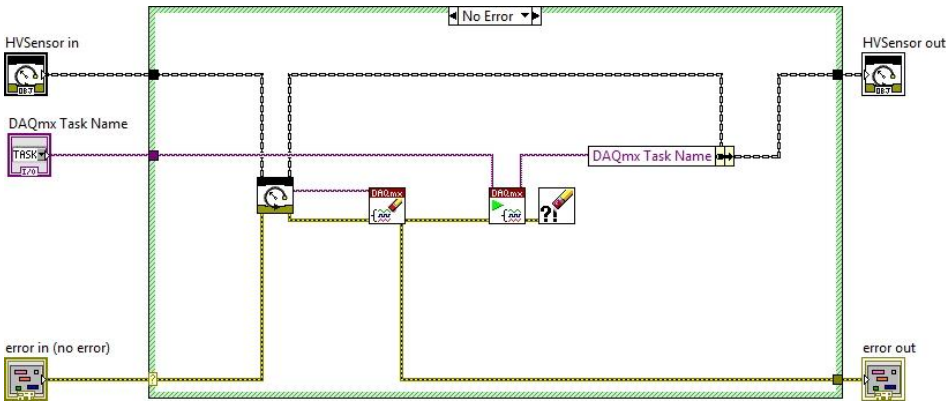


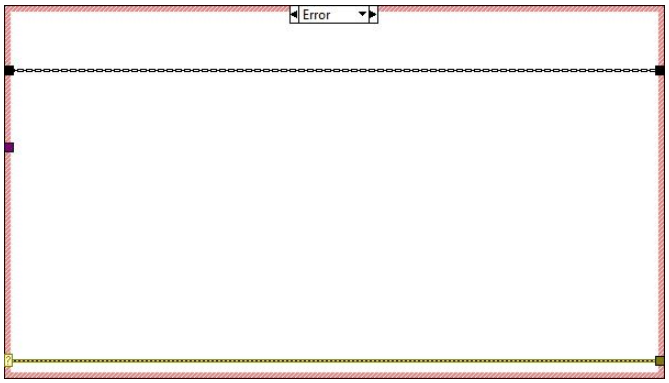




"AccLab Vacuum.lvlib:HVSensor.lvclass:set Controller Type.vi History"

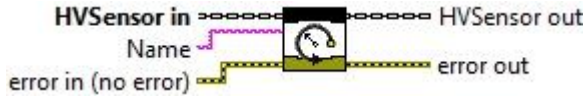
2.7 AccLab Vacuum.lvlib:HVSensor.lvclass:**set DAQMx Task.vi**

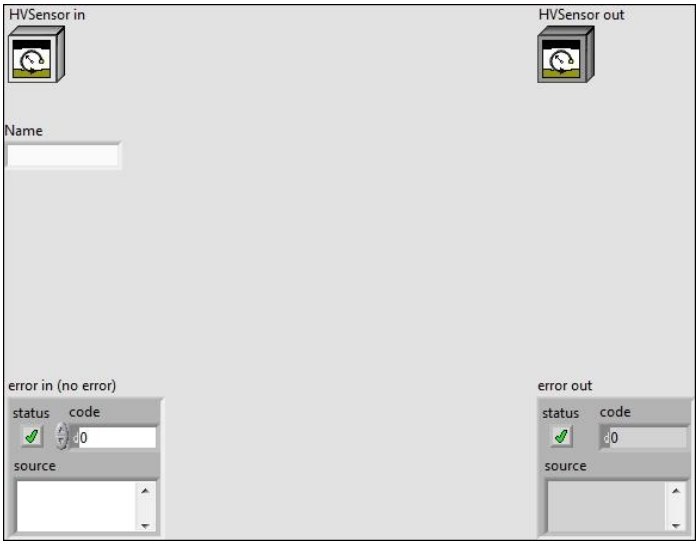


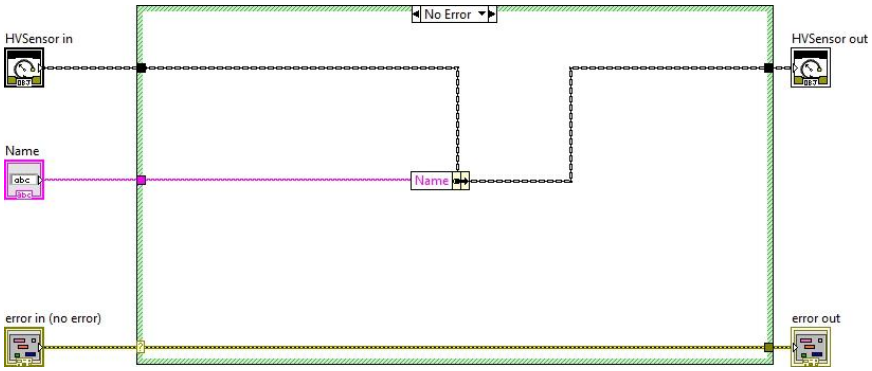


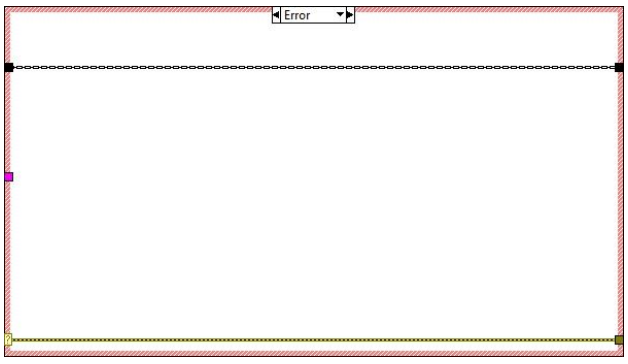
"AccLab Vacuum.lvlib:HVSensor.lvclass:set DAQMx Task.vi History"

2.8 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Name.vi**



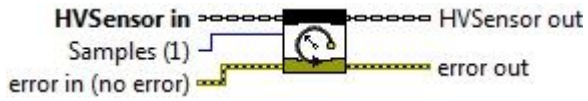


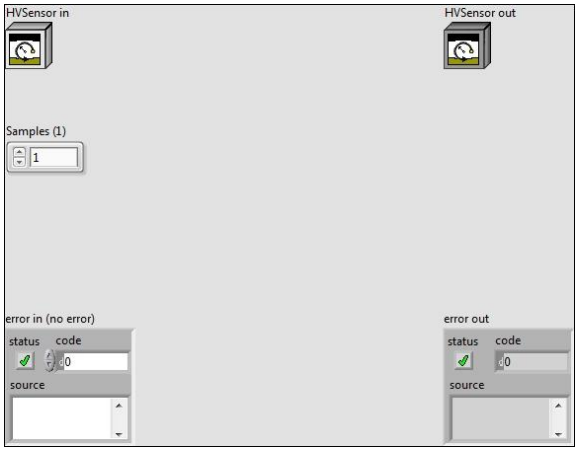


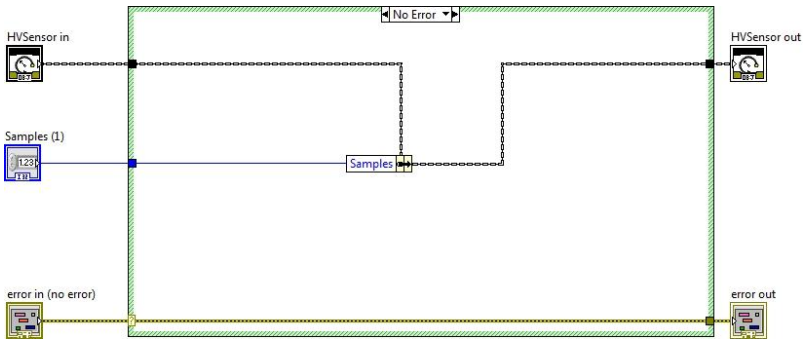


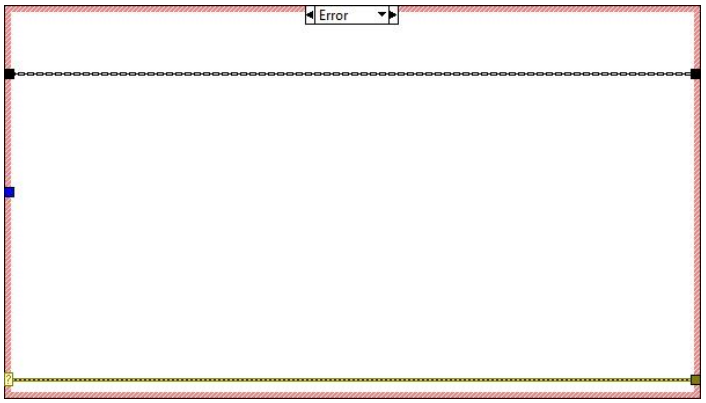
"AccLab Vacuum.lvlib:HVSensor.lvclass:set Name.vi History"

2.9 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Samples.vi**



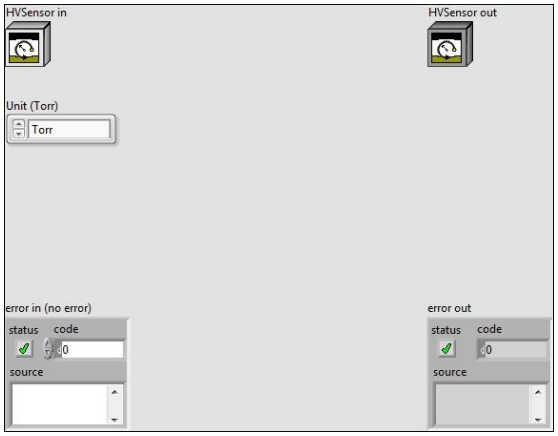


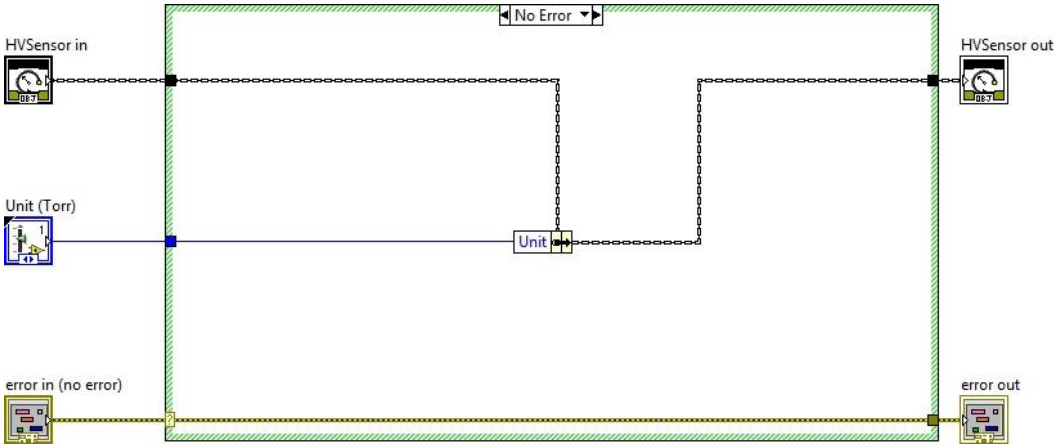


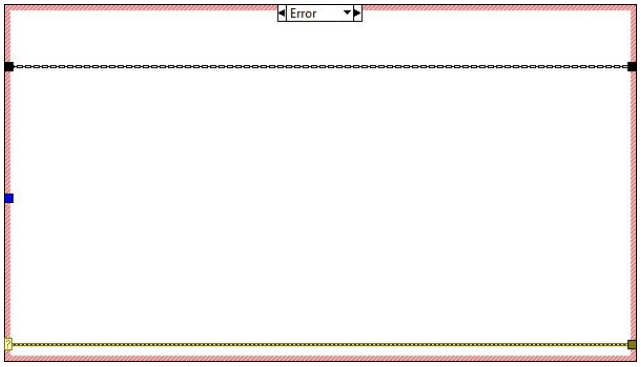


"AccLab Vacuum.lvlib:HVSensor.lvclass:set Samples.vi History"

2.10 AccLab Vacuum.lvlib:HVSensor.lvclass:**set Unit.vi**





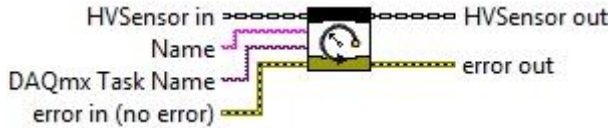


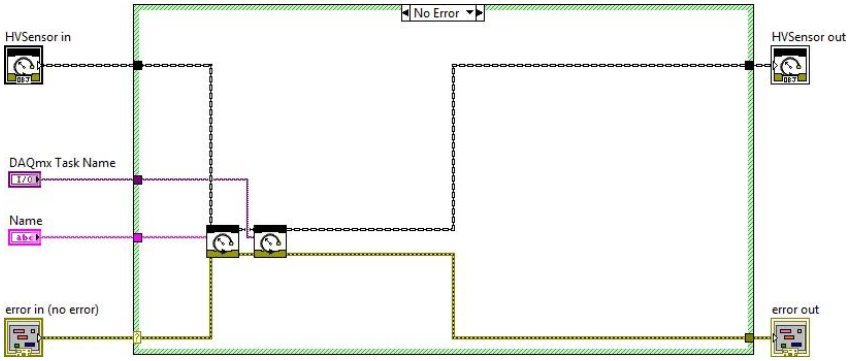
"AccLab Vacuum.lvlib:HVSensor.lvclass:set Unit.vi History"

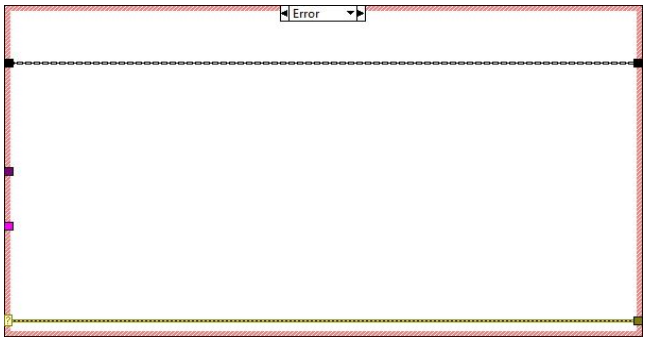
3 Init/kill-aliohjelmat

3.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**init.vi**

Initialisze Name value and DAQMx Task Name value

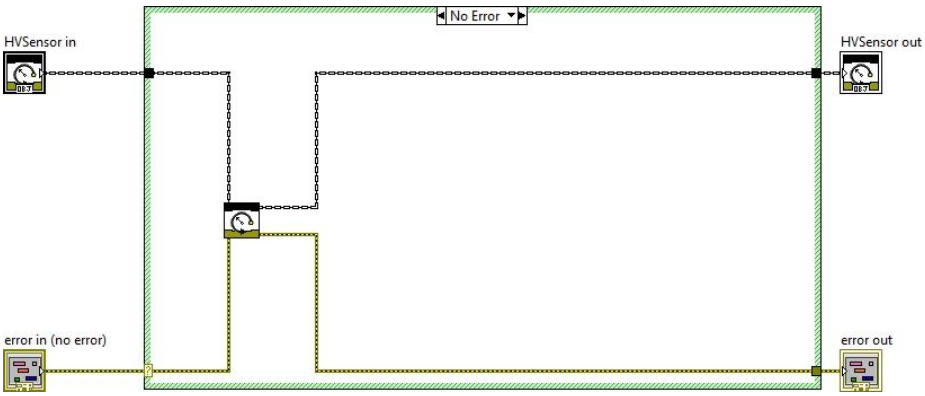


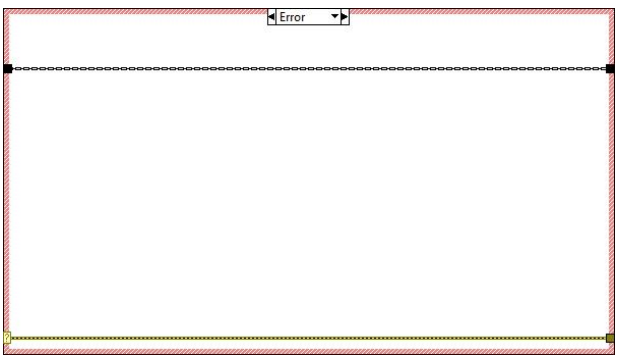




3.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**kill.vi**

Stops the DAQmx task.





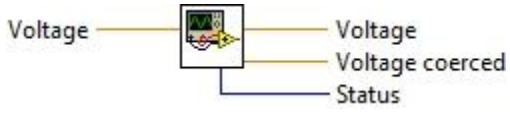
4 Sensor out status –aliohjelmat

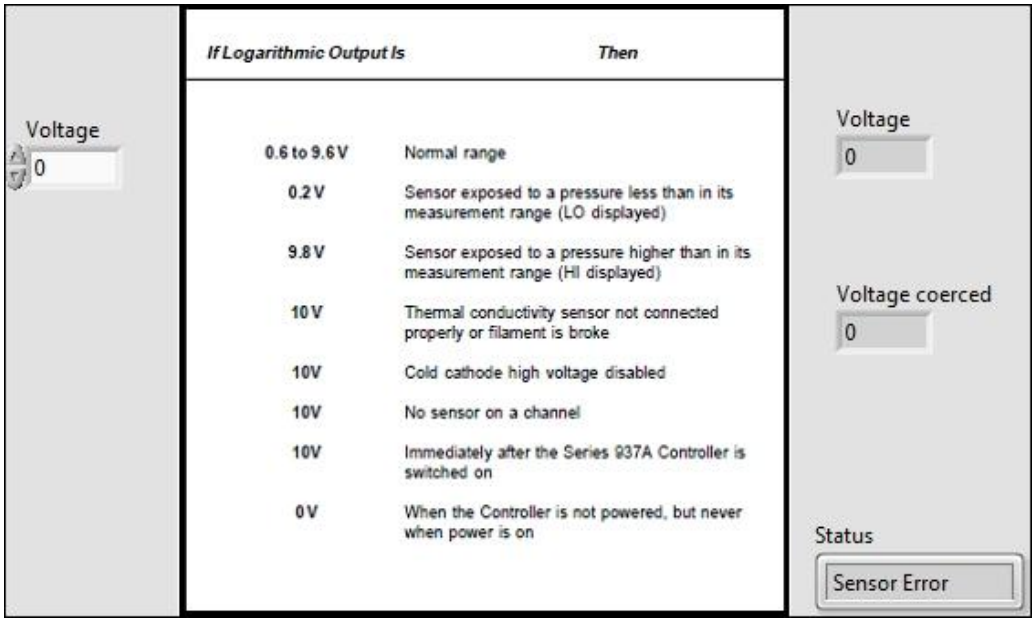
4.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**sensor Output Status (HPS 937A).vi**

For HPS 937A controller.

Returns Sensor Output Status by comparing Voltage to sensors voltage range specifications.

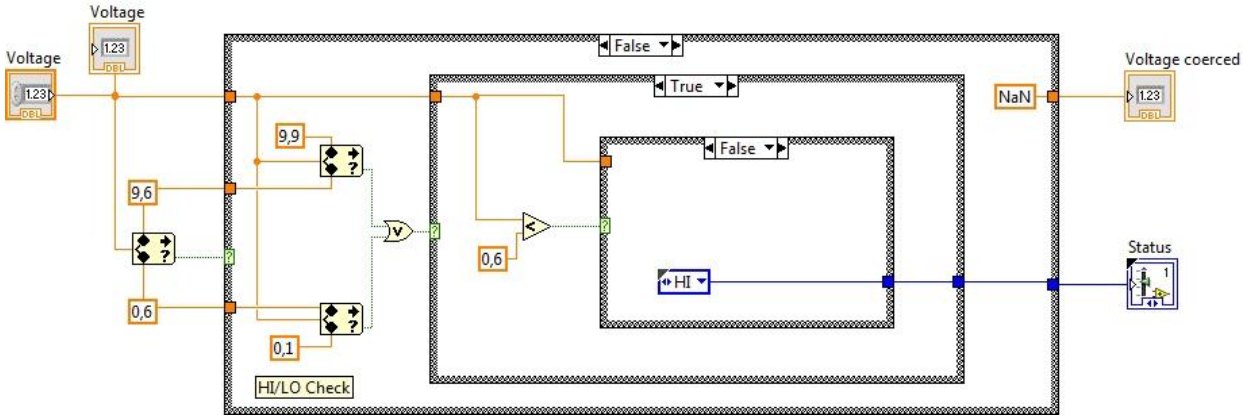
Voltage coerced value is NaN if **Status** is other than "Normal Range".

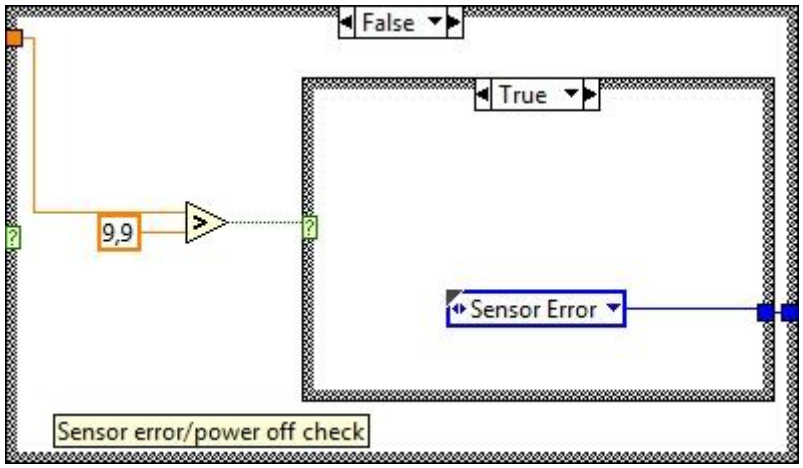


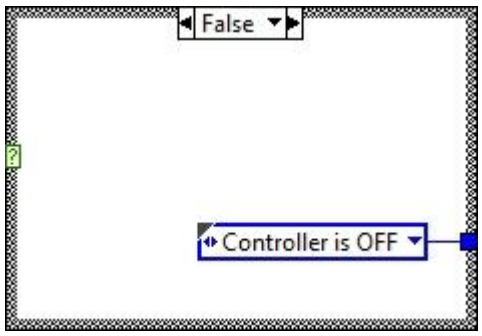


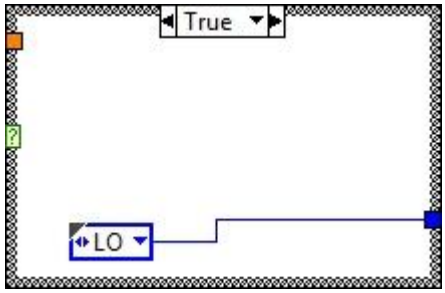
Voltage
Status
Voltage
Voltage coerced

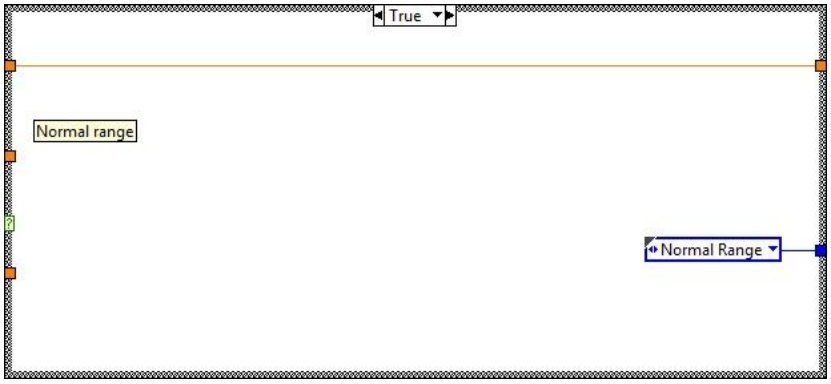
NaN if **Status** is not "Normal Range".


Voltage
Voltage
False
True
False
NaN
Voltage coerced
9,9
9,6
0,6
0,6
HI
Status
0,1
HI/LO Check


False
True
9,9
Sensor Error
Sensor error/power off check


False
Controller is OFF


True
LO

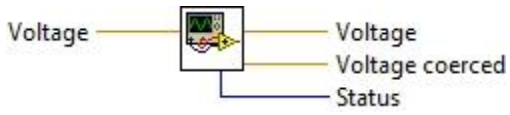


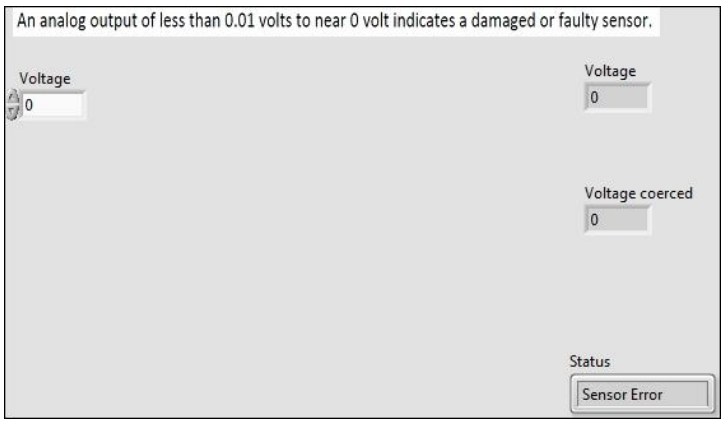
4.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**sensor Output Status (VGC-301A).vi**

For VGC-301A controller.

Returns Sensor Output Status by comparing Voltage to sensors voltage range specifications.

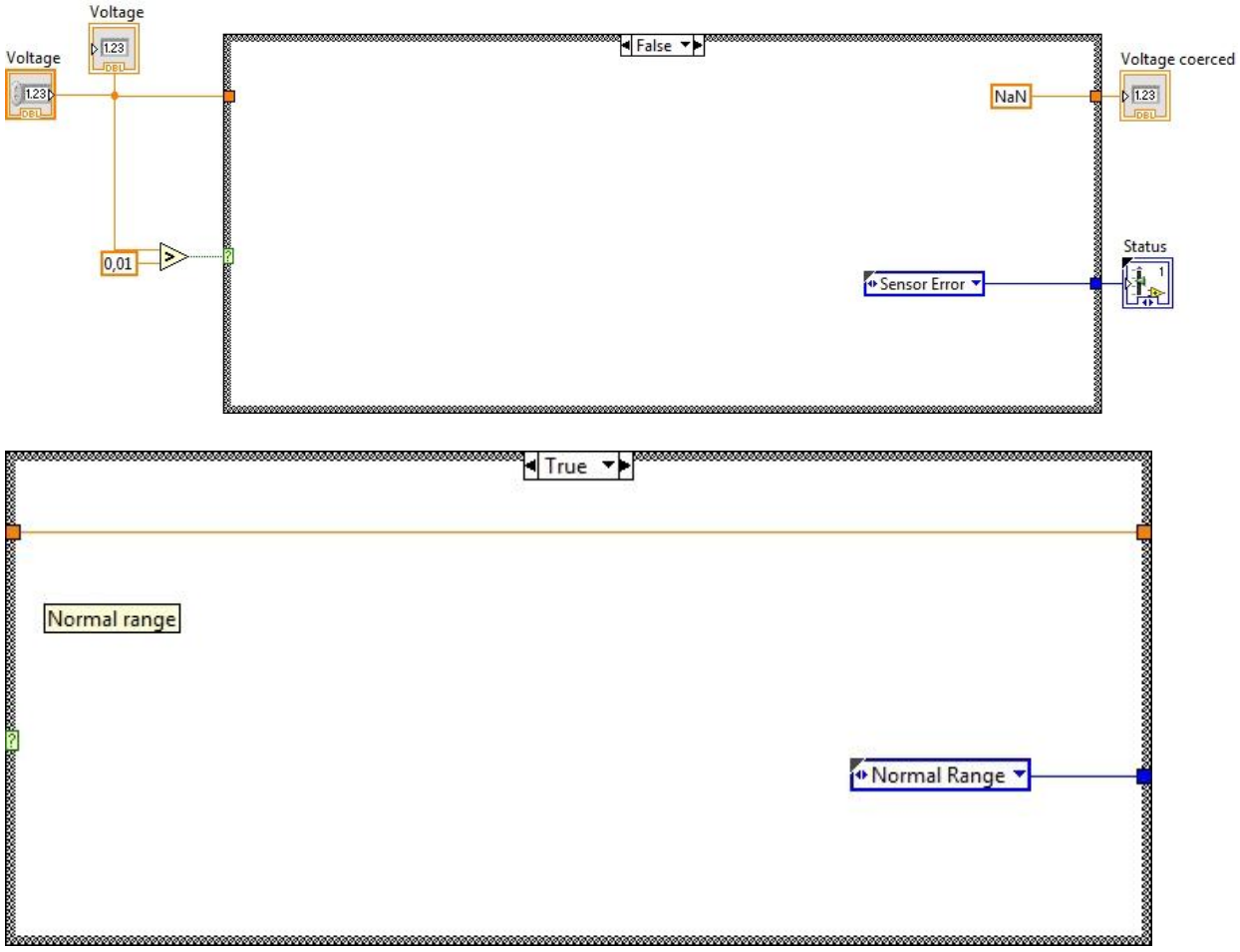
Voltage coerced value is NaN if **Status** is other than "Normal Range".





Voltage
Status
Voltage
Voltage coerced

NaN if **Status** is not "Normal Range".



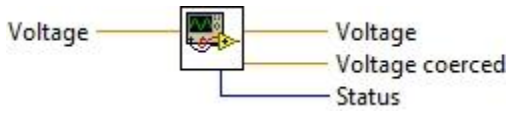
4.3 AccLab Vacuum.lvlib:HVSensor.lvclass:**sensor Output Status (IGM-402).vi**

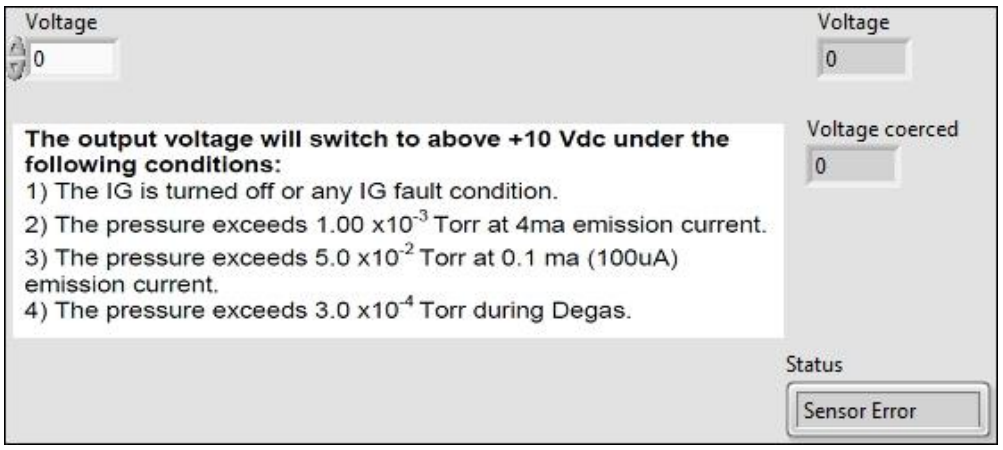
For IGM-402 controller.

Ion gauge Analog Output=IG only(Nitrogen/Air only)

Returns Sensor Output Status by comparing Voltage to sensors voltage range specifications.

Voltage coerced value is NaN if **Status** is other than "Normal Range".





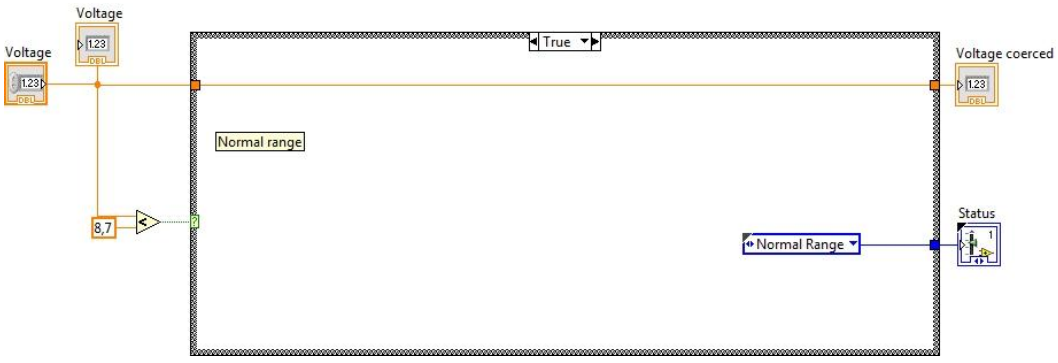
Voltage

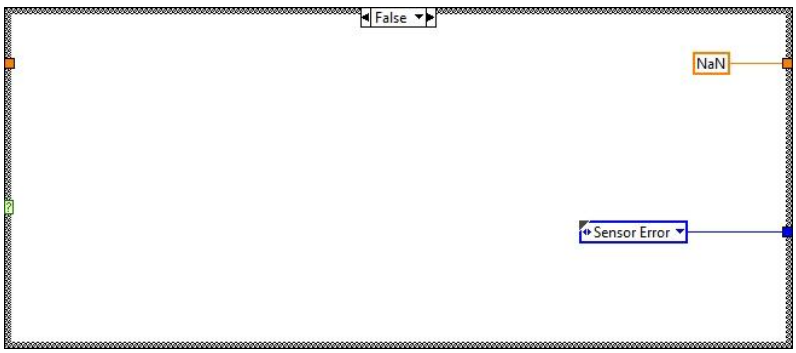
Status

Voltage

Voltage coerced

NaN if **Status** is not "Normal Range".





5 sub voltage to pressure –aliohjelmat

5.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**sub Voltage to Pressure (HPS 937A).vi**

For HPS 937A controller.

Returns voltage to pressure canversion product **Pressure** according to **Unit**

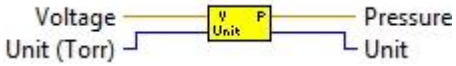
Variants:

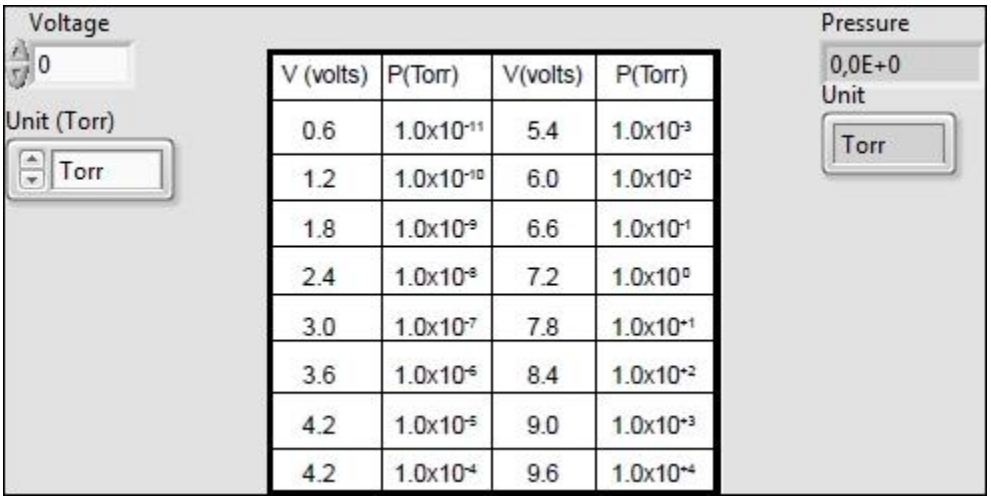
microns

Torr

Pa

mbar





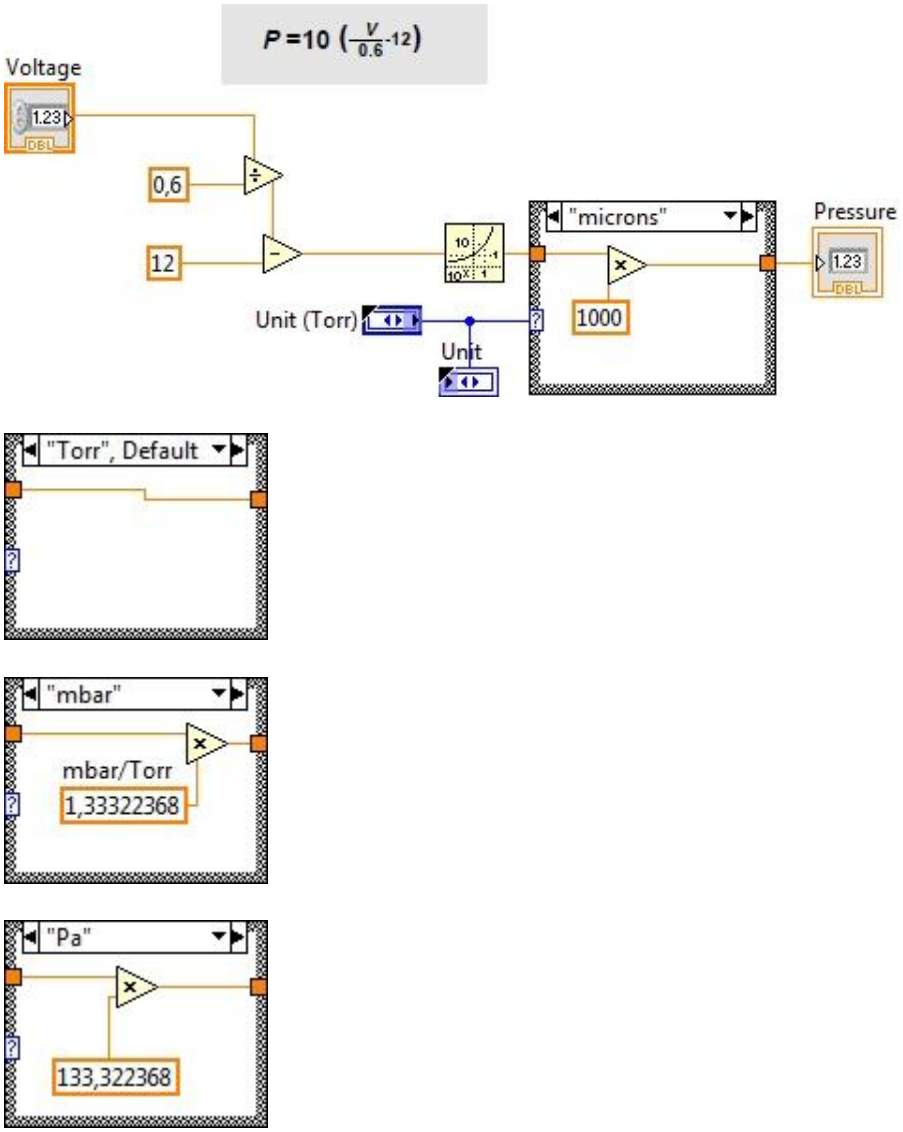
V (volts)	P(Torr)	V(volts)	P(Torr)
0.6	$1.0x10^{-11}$	5.4	$1.0x10^{-3}$
1.2	$1.0x10^{-10}$	6.0	$1.0x10^{-2}$
1.8	$1.0x10^{-9}$	6.6	$1.0x10^{-1}$
2.4	$1.0x10^{-8}$	7.2	$1.0x10^{0}$
3.0	$1.0x10^{-7}$	7.8	$1.0x10^{+1}$
3.6	$1.0x10^{-6}$	8.4	$1.0x10^{+2}$
4.2	$1.0x10^{-5}$	9.0	$1.0x10^{+3}$
4.2	$1.0x10^{-4}$	9.6	$1.0x10^{+4}$

Voltage

Unit (Torr)

Pressure

Unit



5.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**sub Voltage to Pressure (IGM-402).vi**

For IGM-402 controller.

analog output (user selectable)=log linear 1 to 8 Vdc, 1V/decade

Returns voltage to pressure canversion product **Pressure** according to **Unit**

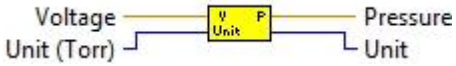
Variants:

microns

Torr

Pa

mbar

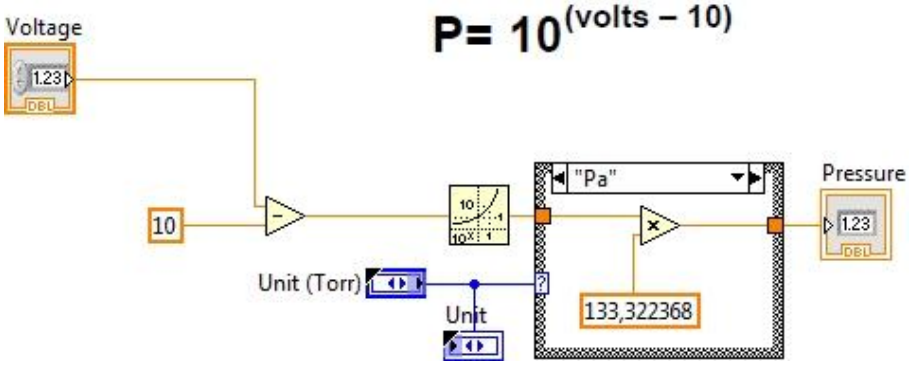


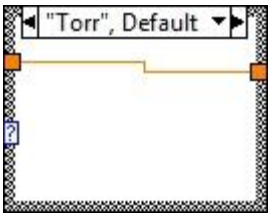
Voltage
0
Unit (Torr)
Torr

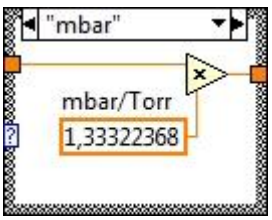
Pressure (Torr)	Voltage
1.0E-10	0.0
1.0E-09	1.0
1.0E-08	2.0
1.0E-07	3.0
1.0E-06	4.0
1.0E-05	5.0
1.0E-04	6.0
1.0E-03	7.0
1.0E-02	8.0
5.0E-02	8.698
Filament if off	≥10

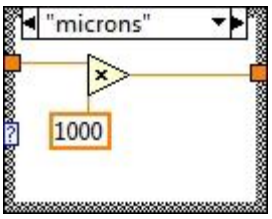
Pressure
0,0E+0
Unit
Torr

Voltage
Unit (Torr)
Pressure
Unit









5.3 AccLab Vacuum.lvlib:HVSensor.lvclass:**sub Voltage to Pressure (VGC-301A).vi**

For VGC-301A controller.

analog output (user selectable)=log linear 1 to 8 Vdc, 1V/decade

Returns voltage to pressure canversion product **Pressure** according to **Unit**

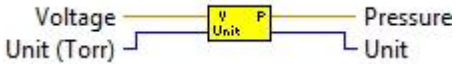
Variants:

microns

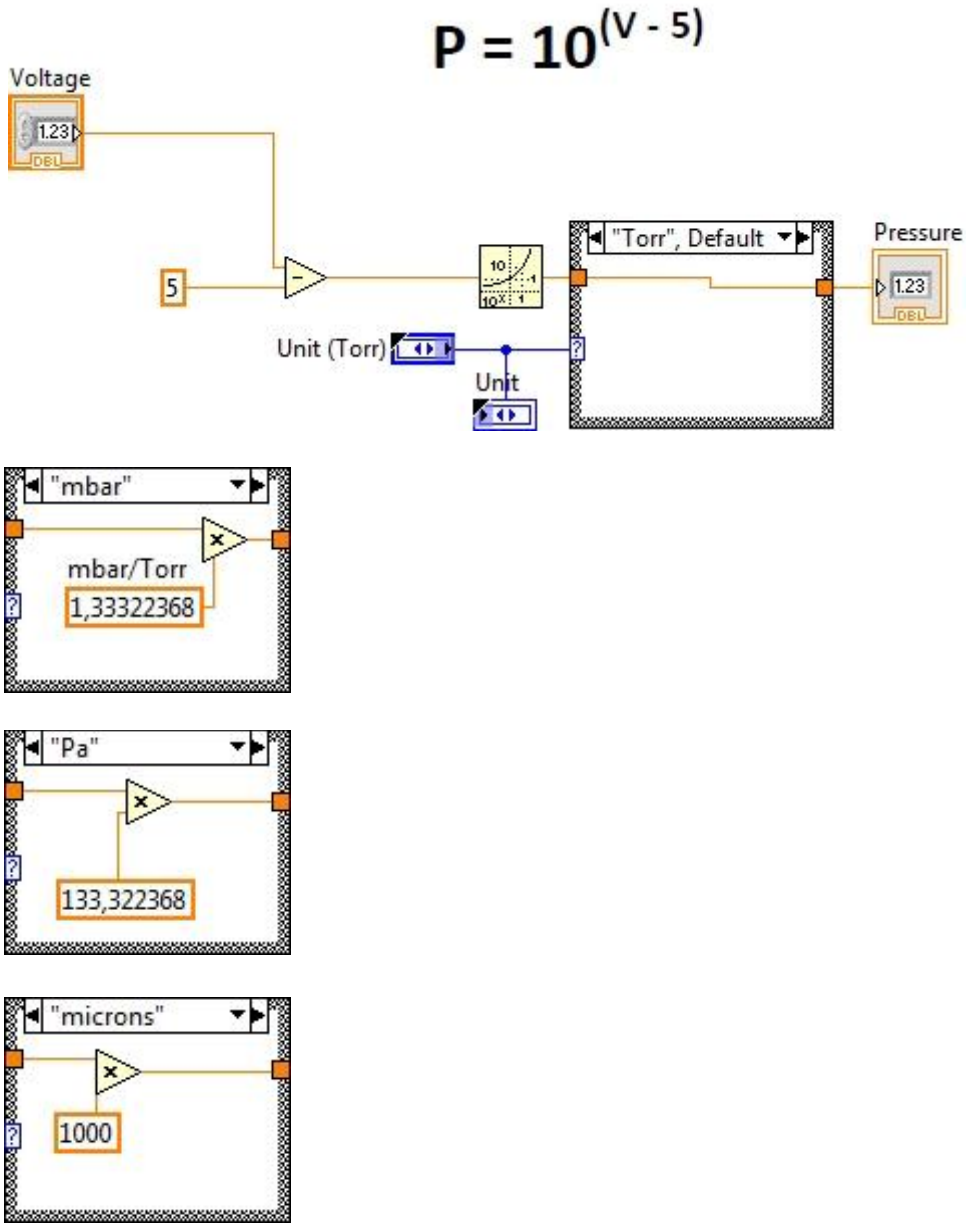
Torr

Pa

mbar

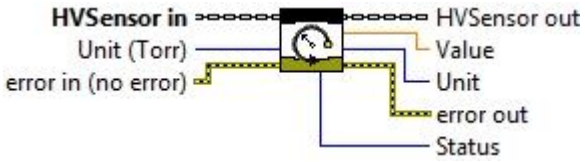


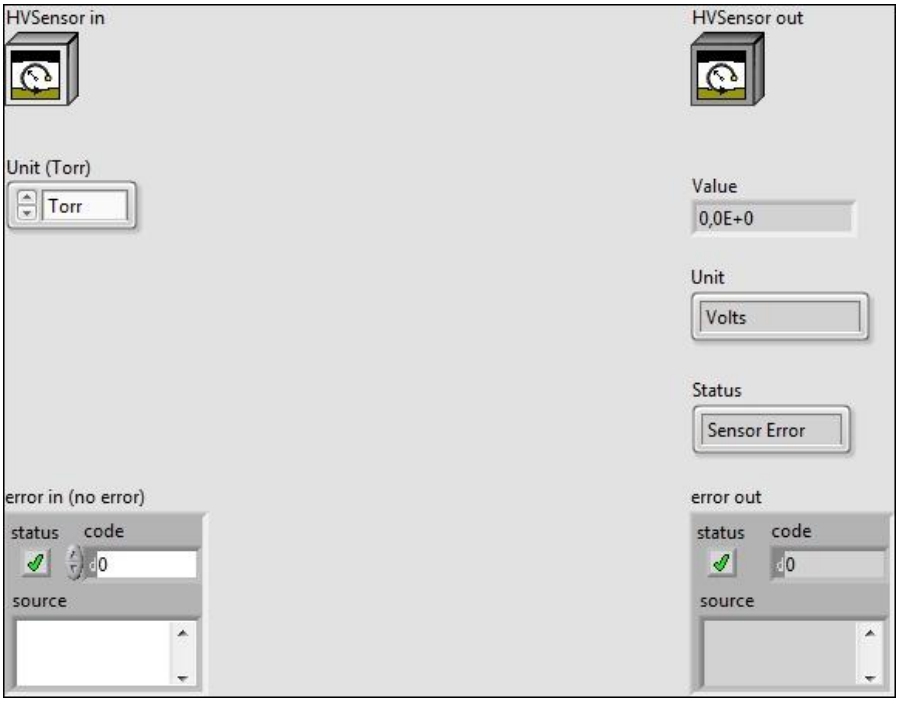
Voltage
Unit (Torr)
Pressure
Unit

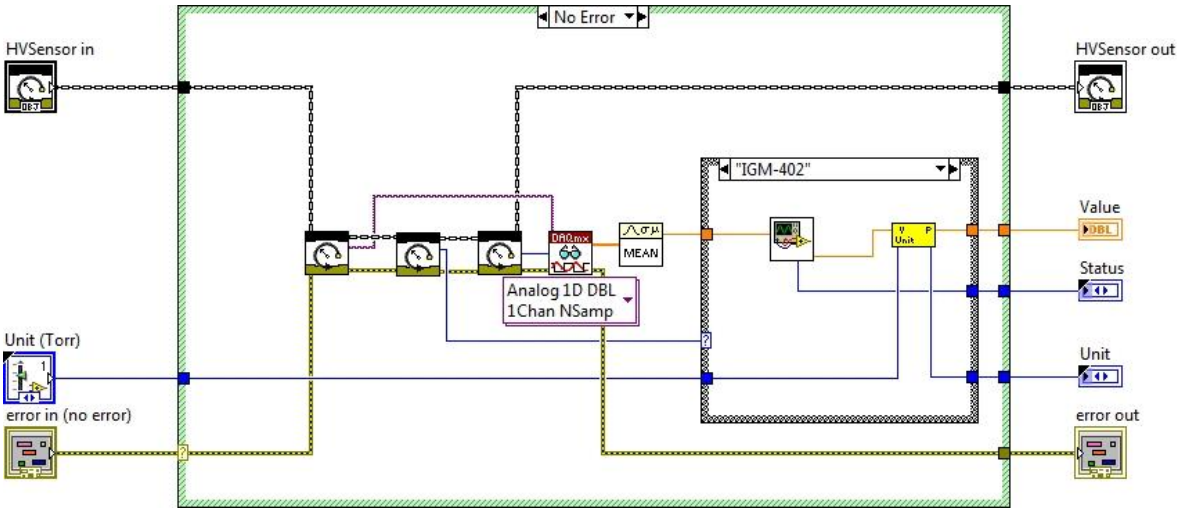


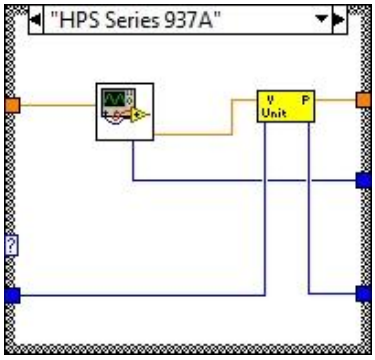
6 AccLab Vacuum.lvlib:HVSensor.lvclass:read Pressure.vi

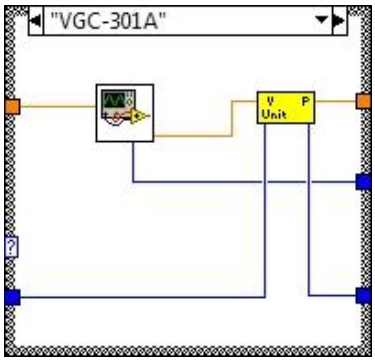
Returns sensors pressure **Value**, **Unit** of the Value and **Sensor Status** according to Unit definition and Sensor Status.

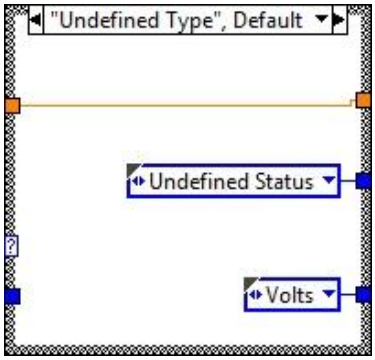

HVSensor in
Unit (Torr)
error in (no error)
HVSensor out
Value
Unit
error out
Status

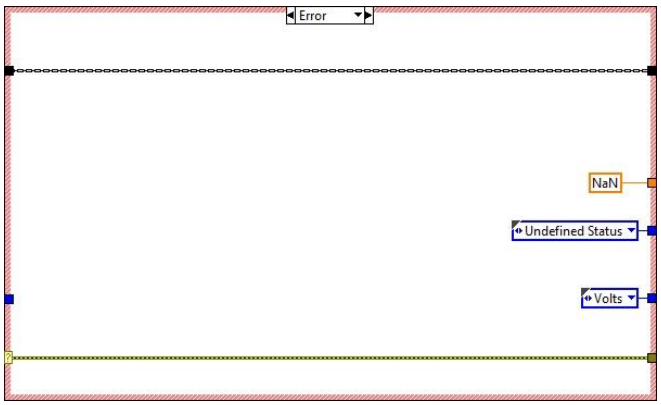

HVSensor in
HVSensor out
Unit (Torr)
Torr
Value
0,0E+0
Unit
Volts
Status
Sensor Error
error in (no error)
status
code
0
source
error out
status
code
0
source


No Error
HVSensor in
HVSensor out
"IGM-402"
Value
Status
Unit
error out
MEAN
Analog 1D DBL
1Chan NSamp
Unit (Torr)
error in (no error)


"HPS Series 937A"
V P
Unit


"VGC-301A"
V P
Unit

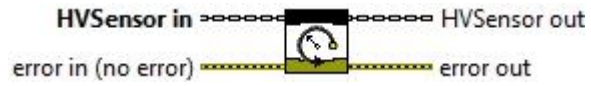

"Undefined Type", Default
Undefined Status
Volts

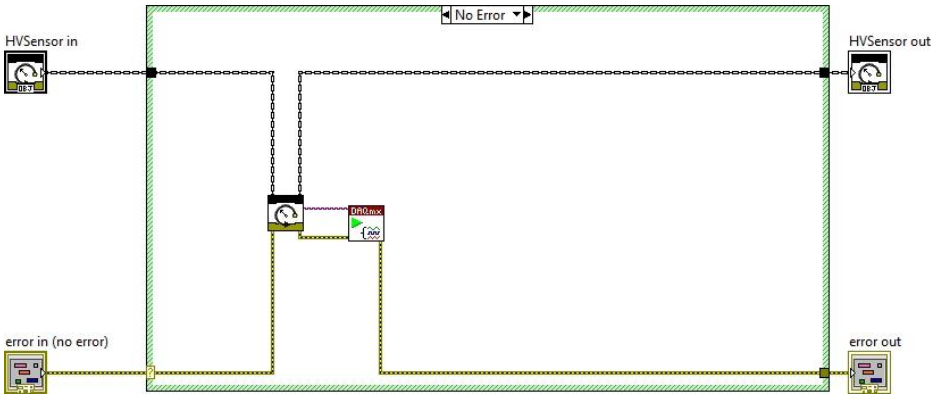

Error
NaN
Undefined Status
Volts

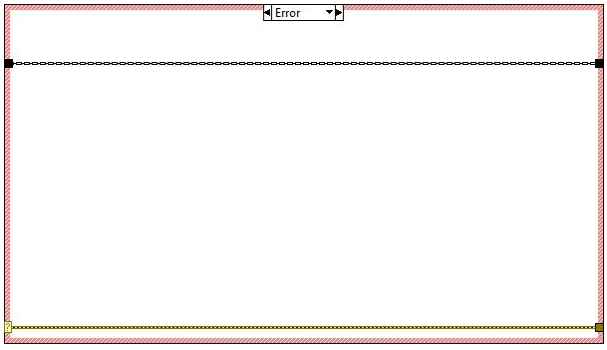
7 Start/Stop DAQMx task –aliohjelmat

7.1 AccLab Vacuum.lvlib:HVSensor.lvclass:**start DAQMx Task.vi**

Starts DAQMx Task.

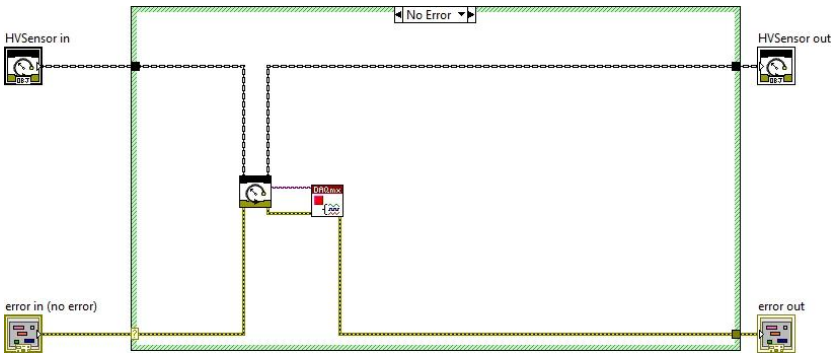


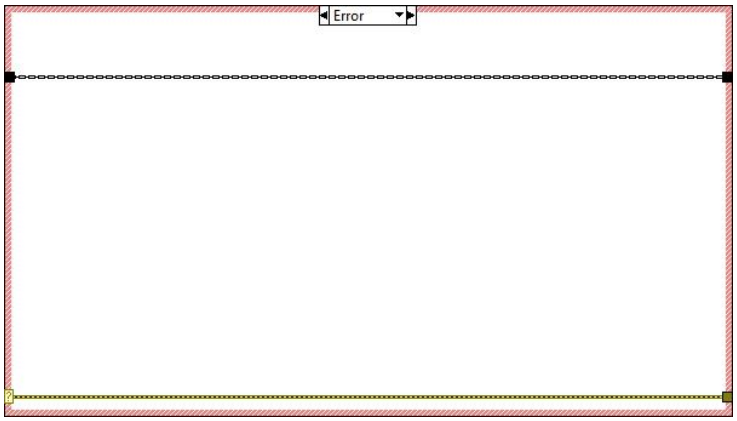




7.2 AccLab Vacuum.lvlib:HVSensor.lvclass:**stop DAQMx Task.vi**

Stops DAQMx Task.





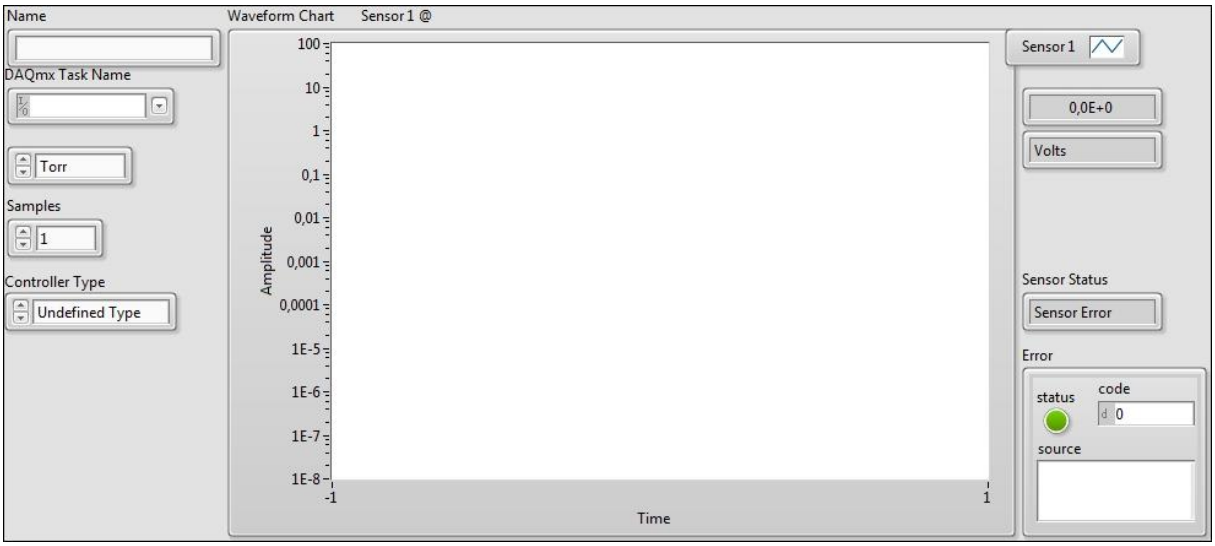
8 Example code –aliohjelmat

8.1 AccLab Vacuum.lvlib:**HVSensor Panel.vi**

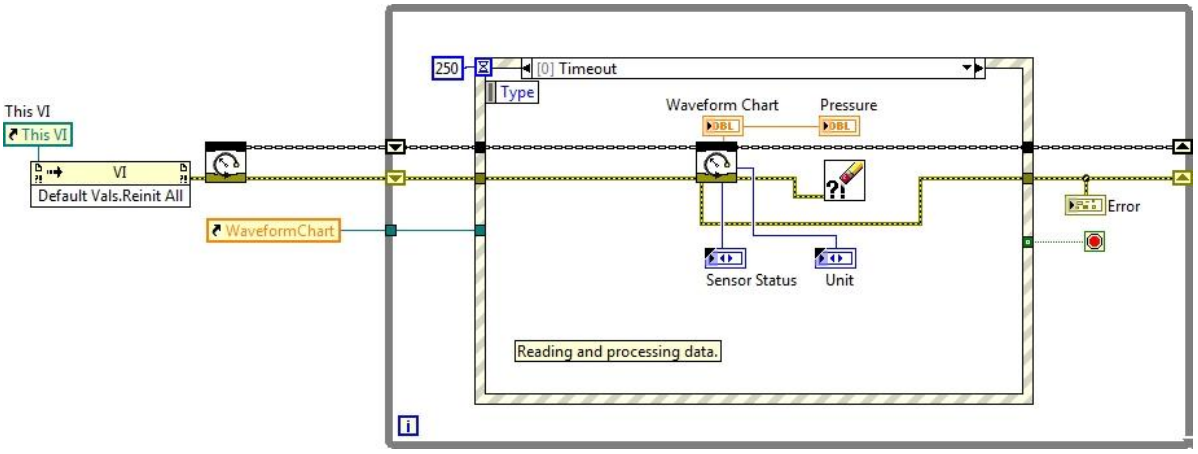
User Interface window.

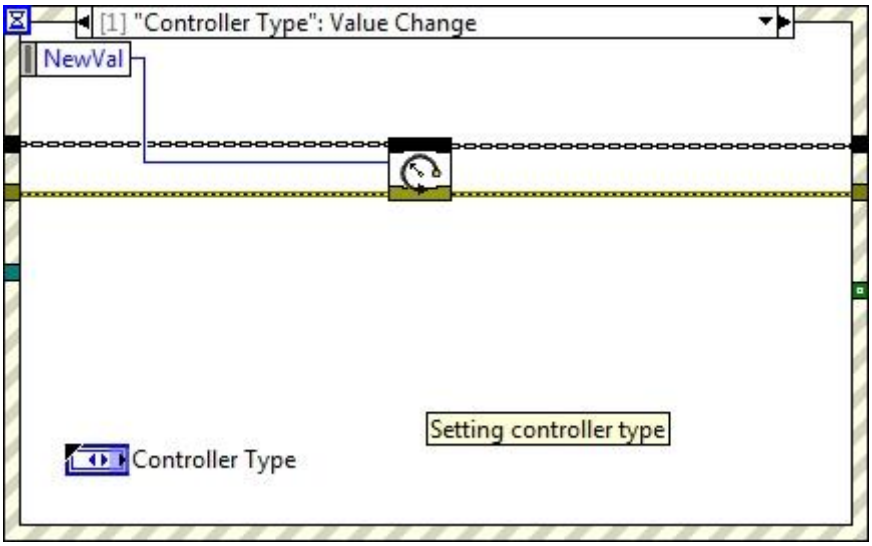
I/O:

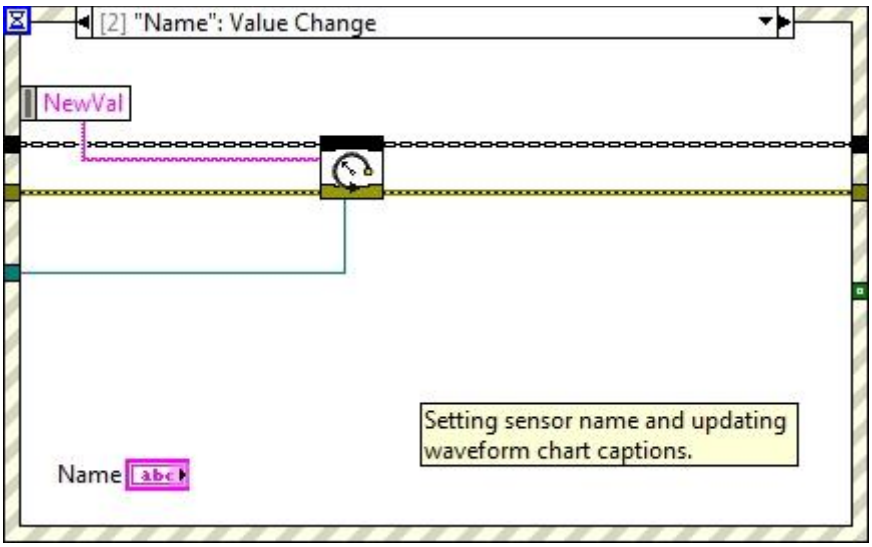
Inputs:	**Outputs:**
Name	Waveform Chart
DAQmx Task	Name Value
Unit	Unit
Samples	Sensor Status
Controller Type	Error

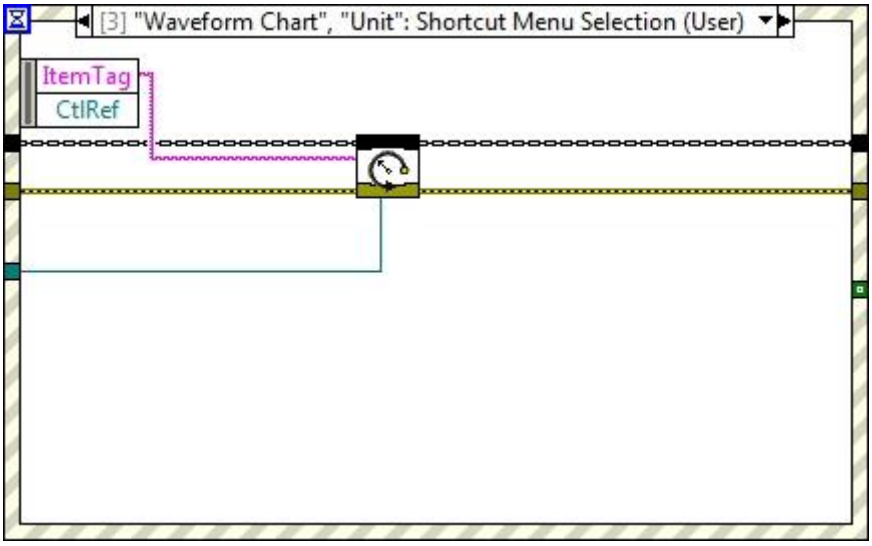


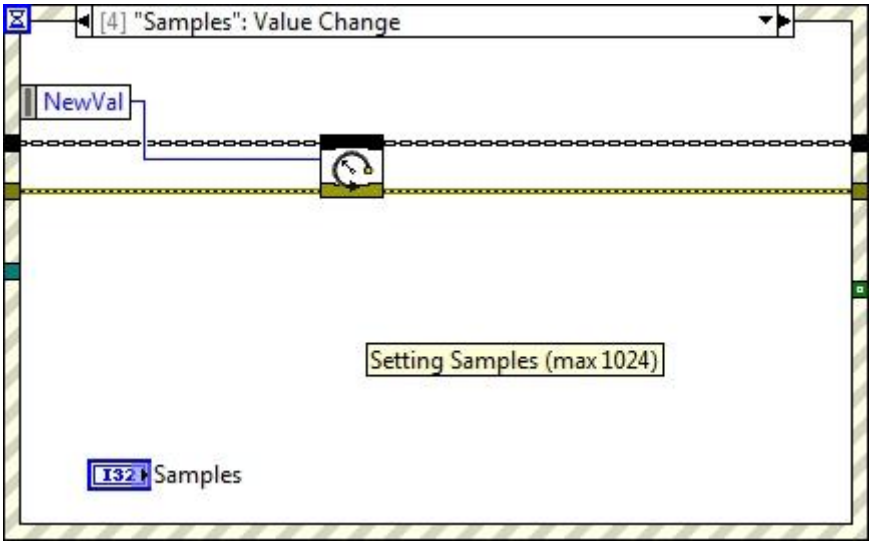
Name
DAQmx Task Name
Unit
Samples
Controller Type
Sensor 1 @ [Waveform Chart]

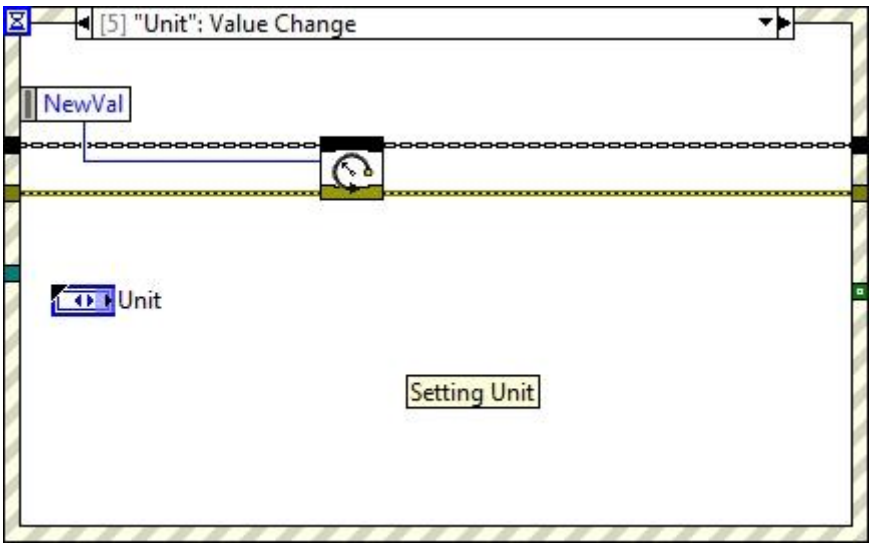

This VI
This VI
VI
Default Vals.Reinit All
WaveformChart
250
[0] Timeout
Type
Waveform Chart
Pressure
Sensor Status
Unit
Error
Reading and processing data.

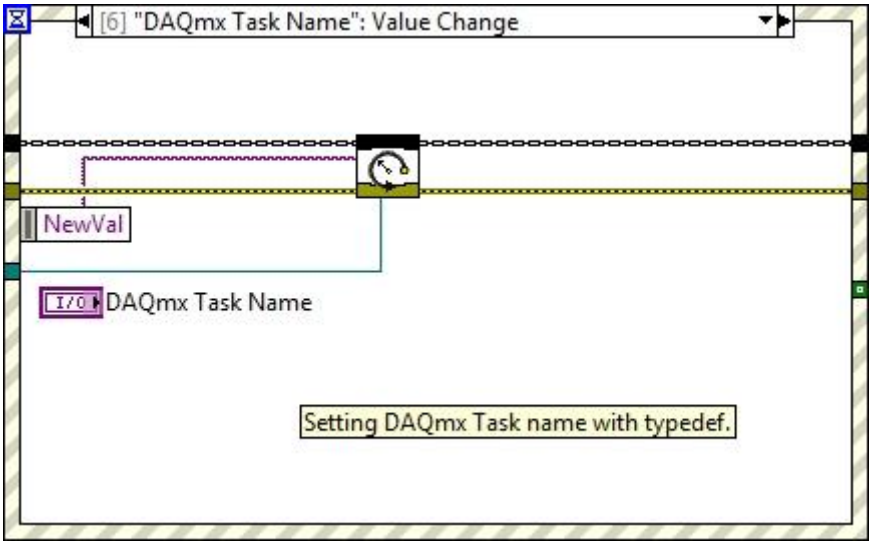

[1] "Controller Type": Value Change
NewVal
Setting controller type
Controller Type

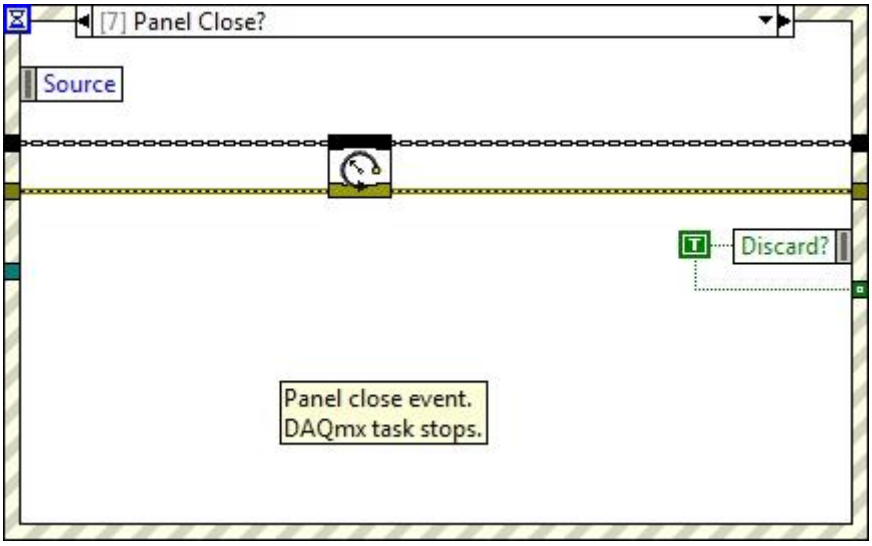

[2] "Name": Value Change
NewVal
Setting sensor name and updating waveform chart captions.
Name

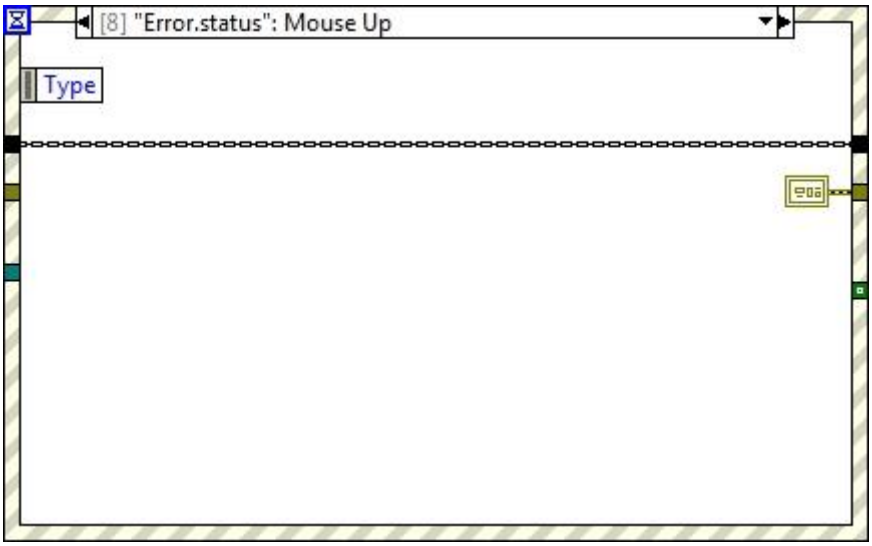

[3] "Waveform Chart", "Unit": Shortcut Menu Selection (User)
ItemTag
CtlRef


[4] "Samples": Value Change
NewVal
Setting Samples (max 1024)
I32
Samples


[5] "Unit": Value Change
NewVal
Unit
Setting Unit

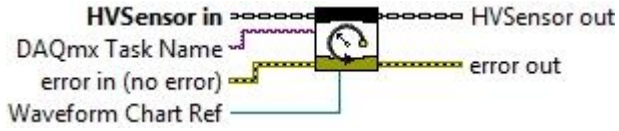

[6] "DAQmx Task Name": Value Change
NewVal
I/O
DAQmx Task Name
Setting DAQmx Task name with typedef.

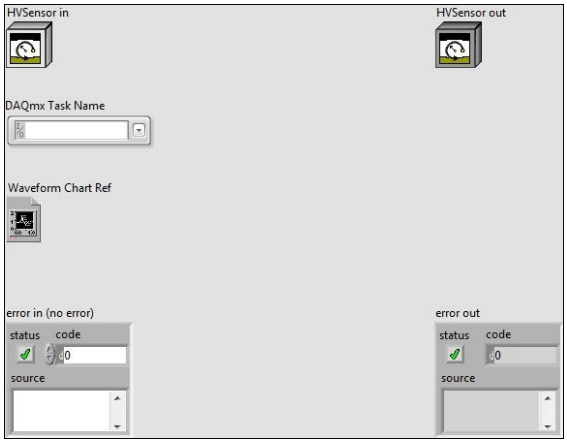

[7] Panel Close?
Source
Discard?
Panel close event.
DAQmx task stops.

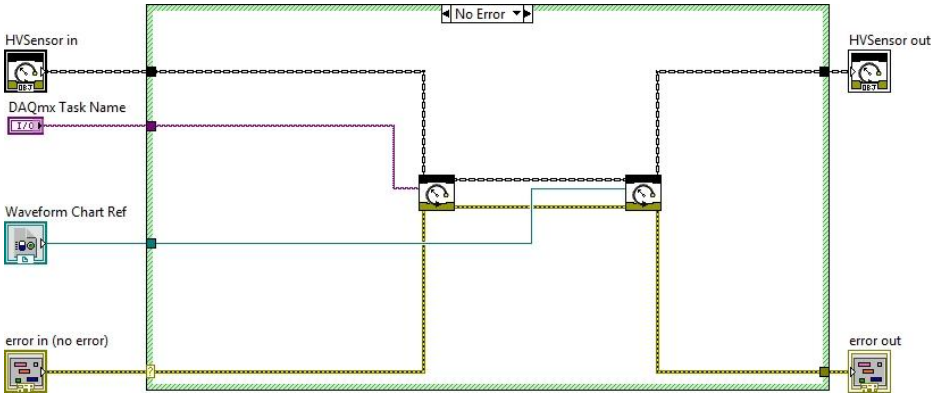

[8] "Error.status": Mouse Up
Type

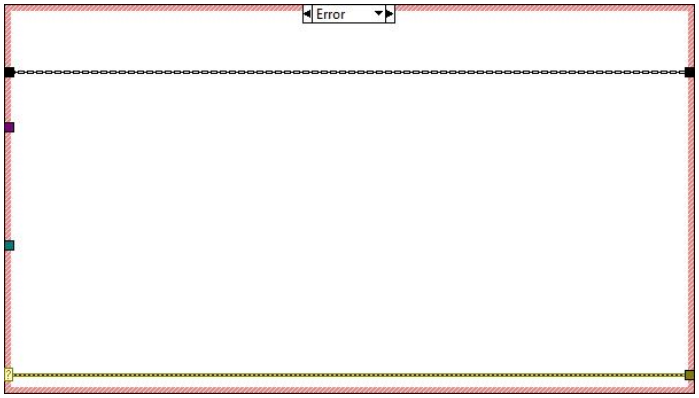
8.2 AccLab Vacuum.lvlib:**handle DAQMx Task Value Change.vi**

Handels DAQmx Task value change event. Sets DAQmx Task Name and updates Waveform Chart caption.



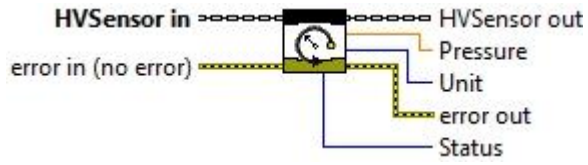


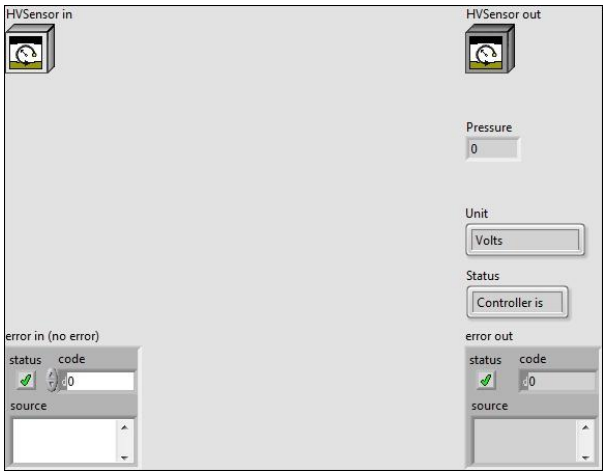


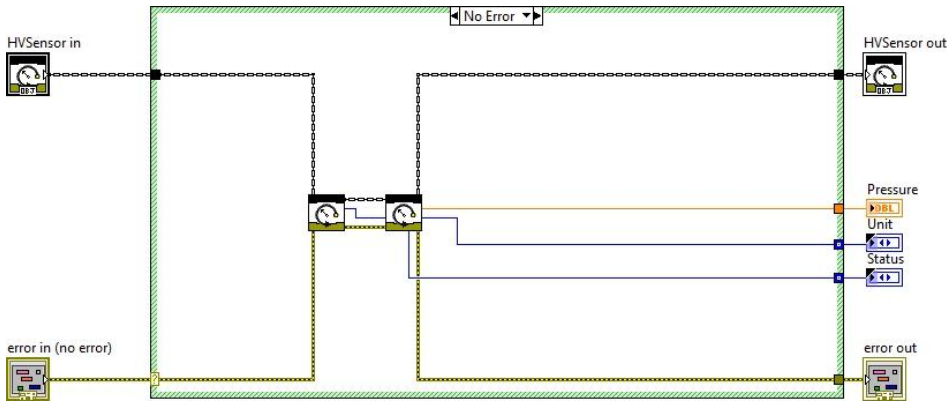


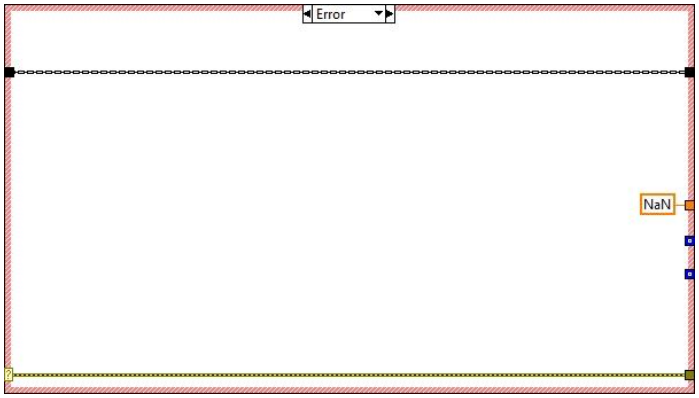
8.3 AccLab Vacuum.lvlib:**handle Timeout.vi**

Returns sensors **Pressure** value, **Unit** of the value and **Sensor Status**.



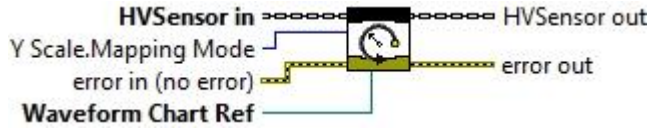


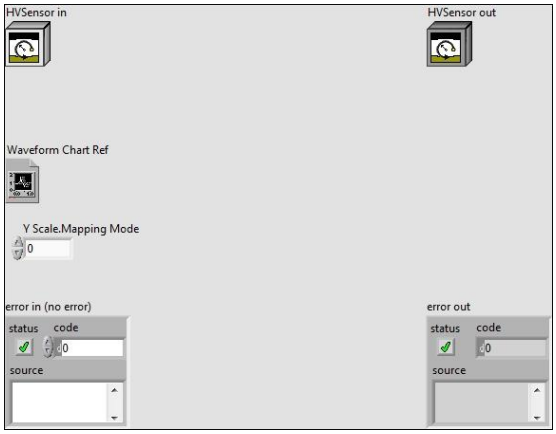


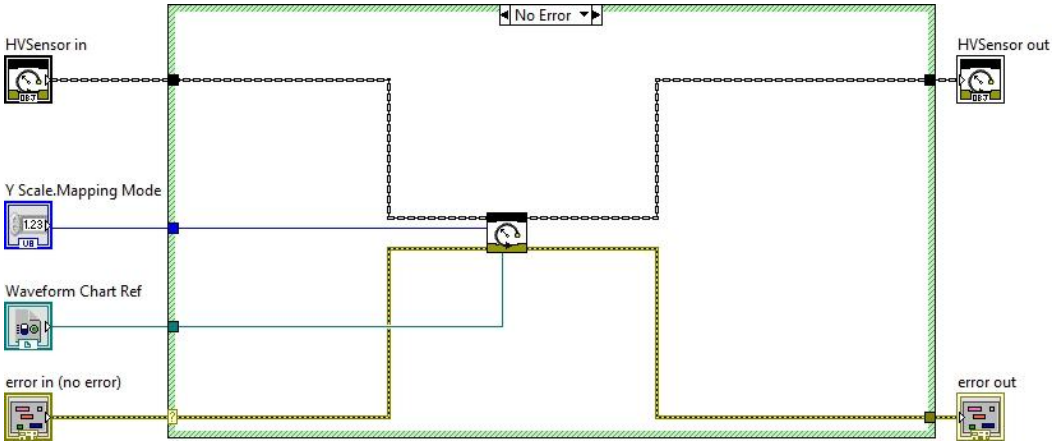


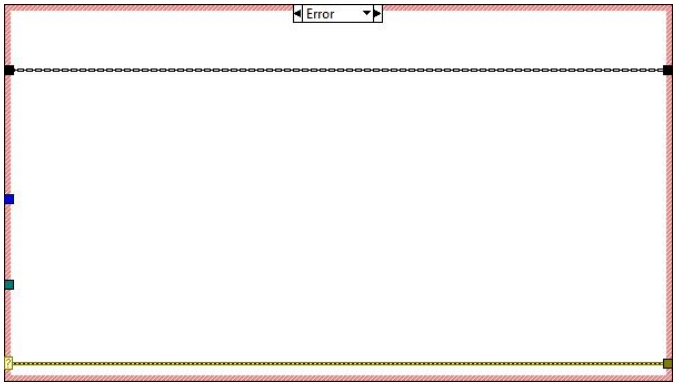
8.4 AccLab Vacuum.lvlib:**handle Waveform Chart mapping .vi**

Handel's Waveform Chart Scale Y mapping. Operates between Linear and Logarithmic scales.



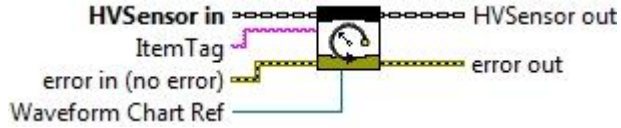


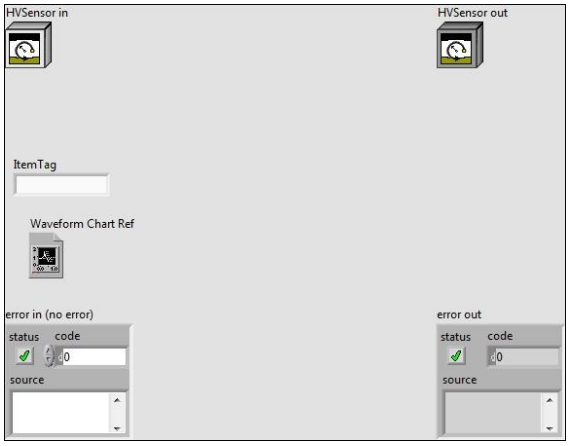


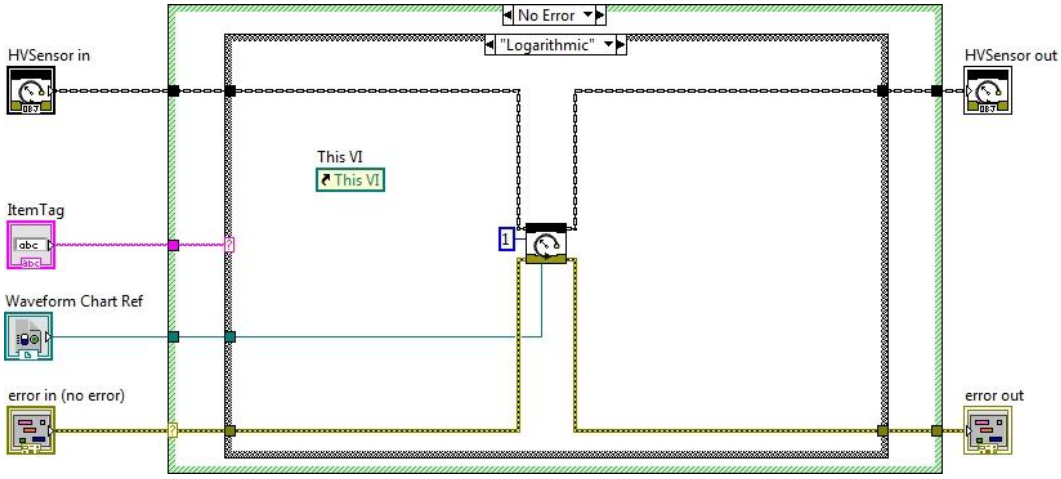


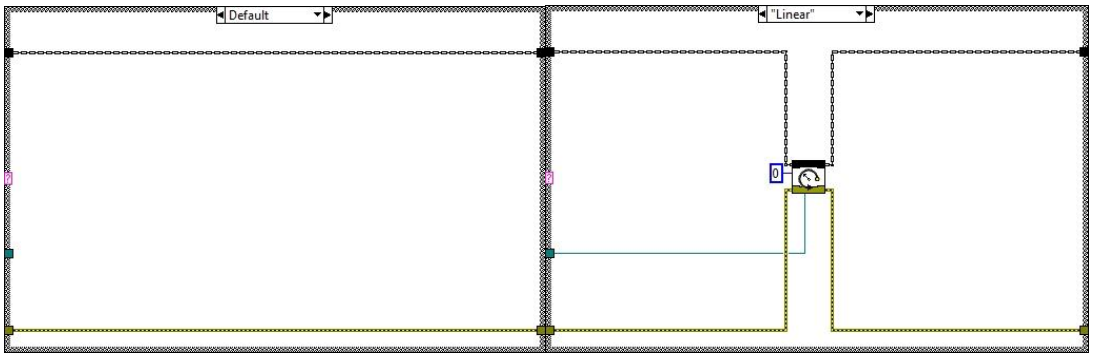
8.5 AccLab Vacuum.lvlib:**handle Waveform Chart Shortcut Menu Selection (User).vi**

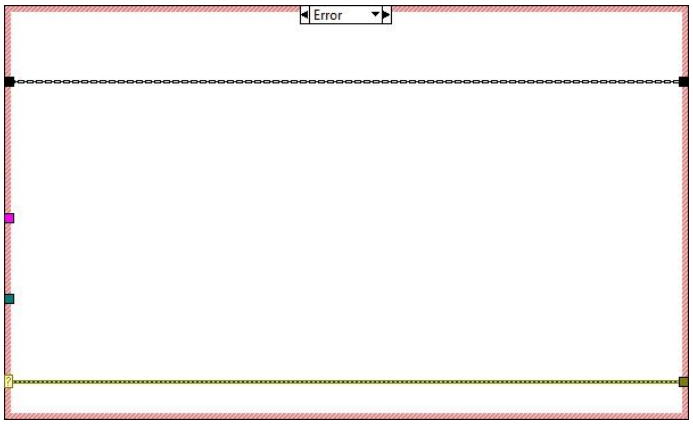
Handles Shortcut Menu for Waveform.





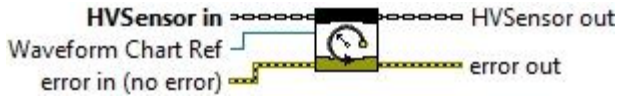


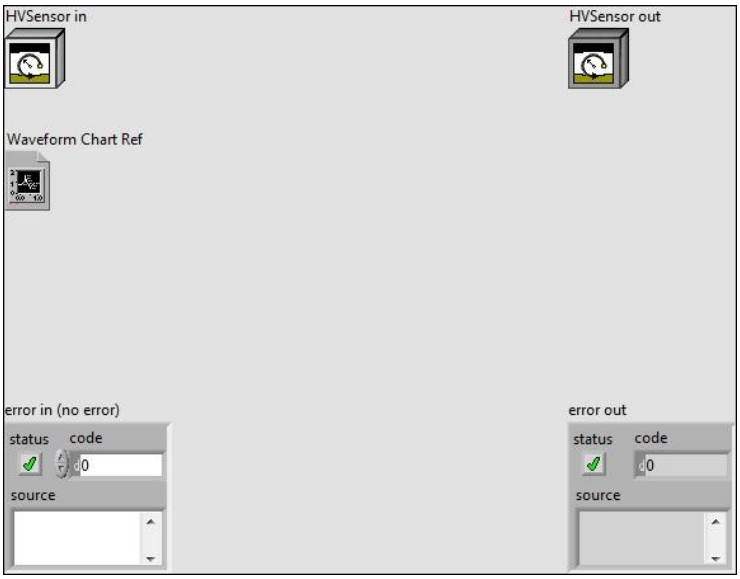


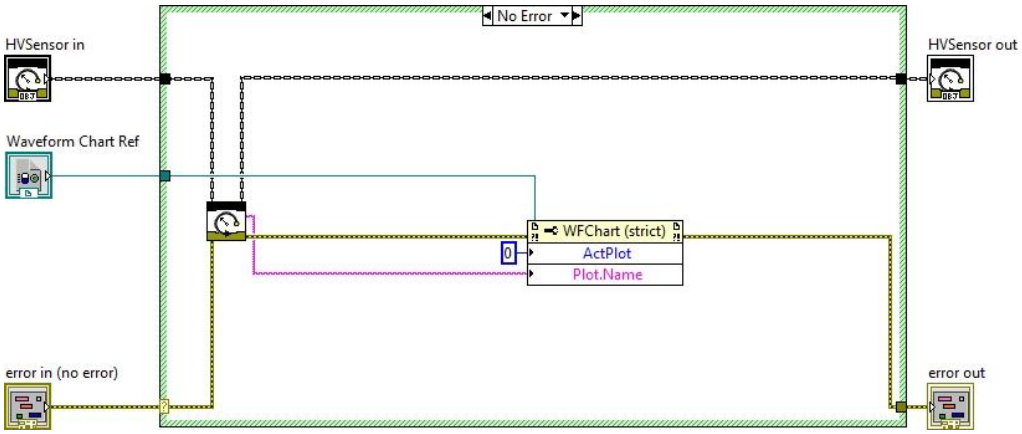


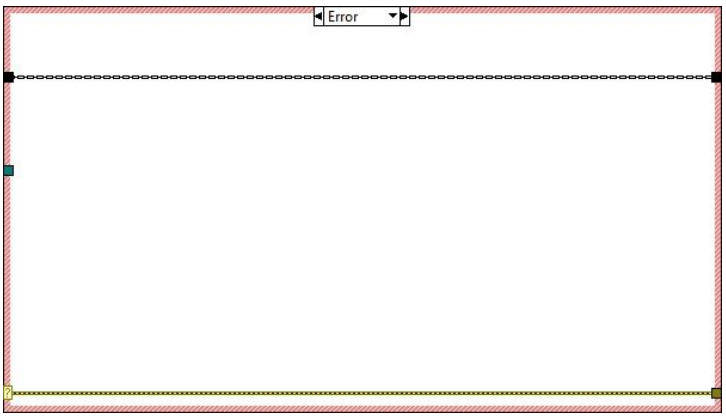
8.6 AccLab Vacuum.lvlib:**update Plot Caption.vi**

Update Plot Caption with **Name**


HVSensor in
Waveform Chart Ref
error in (no error)
HVSensor out
error out

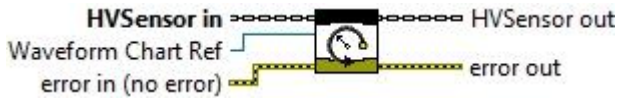

HVSensor in
HVSensor out
Waveform Chart Ref
error in (no error)
status
code
0
source
error out
status
code
0
source

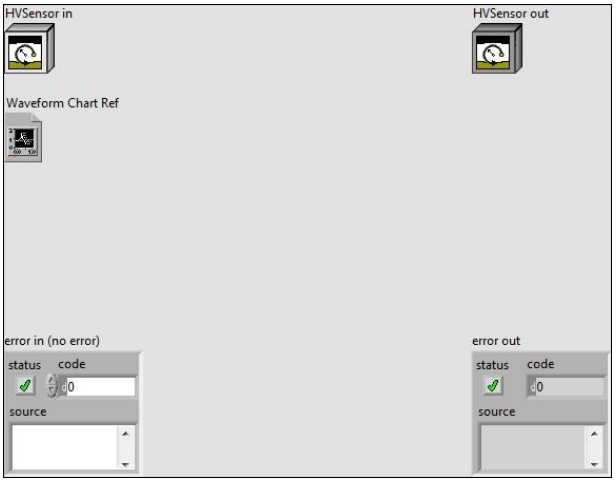

No Error
HVSensor in
HVSensor out
Waveform Chart Ref
WFChart (strict)
0
ActPlot
Plot.Name
error in (no error)
error out

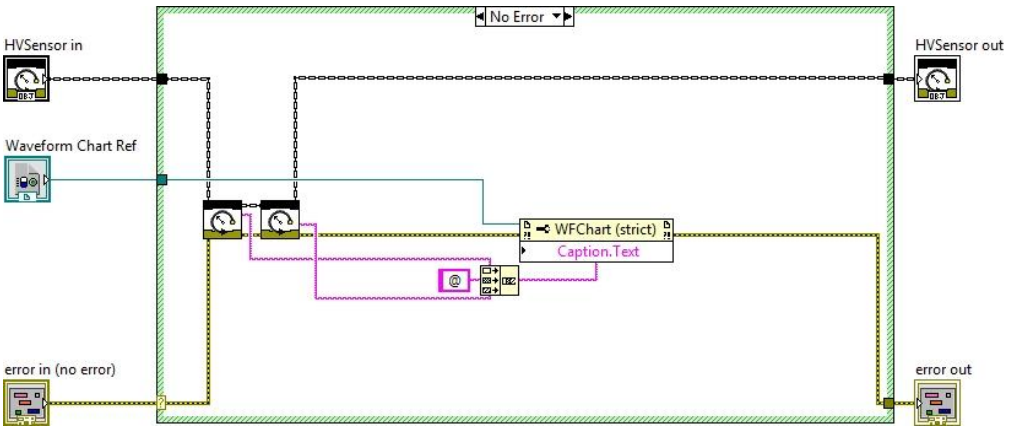

Error

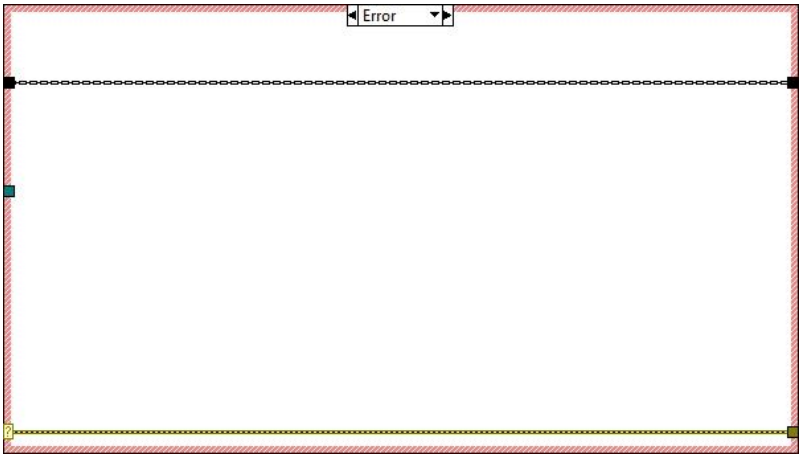
8.7 AccLab Vacuum.lvlib:**update Waveform Chart Caption.vi**

Updates Waveform Chart Caption with **Name** and **DAQmx Task Name**.

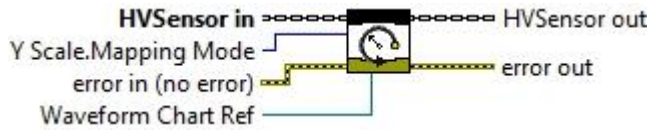


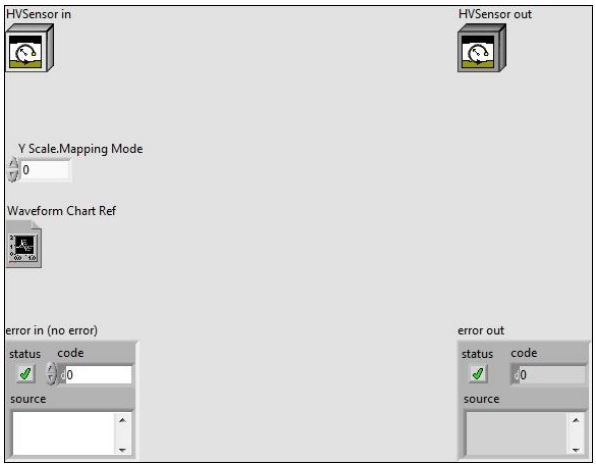


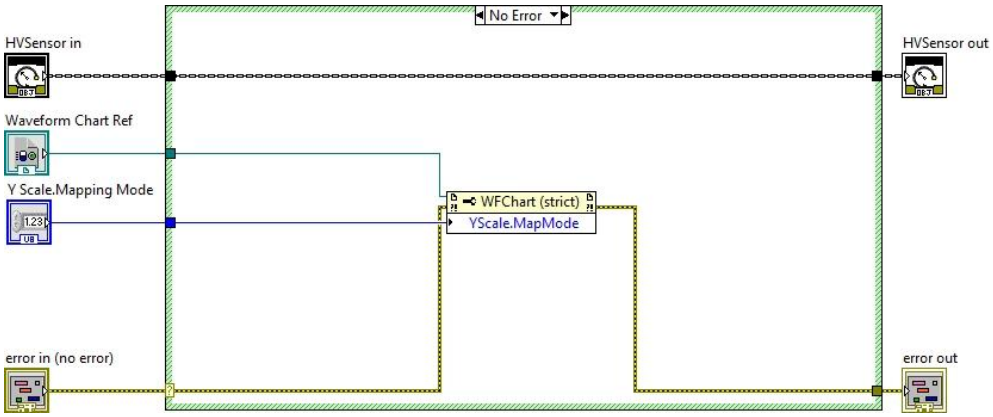


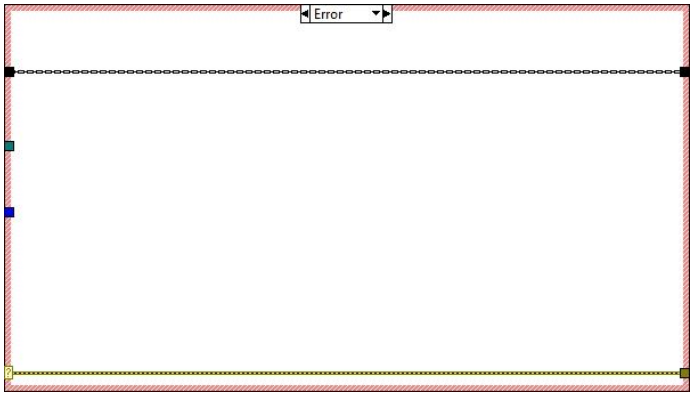


8.8 AccLab Vacuum.lvlib:**update Waveform Chart mapping.vi**









8.9 AccLab Vacuum.lvlib:**handle Sensor Name Value Change.vi**

Handels Sensor Name Value Change event. Sets Sensor Name and updates Waveform Chart caption.