Tietotyypit ja niiden käsittely

Tietotyyppi on tiedon tallennusmuoto, mikä määrittelee, miten ohjelma käsittelee tietoa. AutoLISPissä tietotyyppejä ovat:

- kokonaisluku (englanniksi integer)
- reaaliluku (real)
- merkkijono / teksti (string)
- symbolit ja muuttujat (symbol, variable)
- lista (list)
- tiedosto-osoitin (file descriptor)
- valintajoukko (selection set)
- entiteettinimi / kokonaisuustunniste (entity name)
- VLA-objekti (VLA-object)

Näiden lisäksi on nil, joka tarkoittaa tiedon puutetta*. Muilla kielillä tämä on usein null.

*) Tarkkaan ottaen nil tarkoittaa tyhjää listaa `'()`, mutta käytännössä tämä ei näy. Tästä lisää lista-osiossa.

Luvut: kokonaisluku (integer) ja reaaliluku (real)

Luvut ovat joko kokonaislukuja `5` tai reaalilukuja `5.0`. Lisp muuttaa kokonaisluvun automaattisesti reaaliluvuksi, jos yksi laskutoimituksen luvuista on reaaliluku. Jakolaskut kokonaisluvuilla eivät huomioi tuloksen desimaaleja lainkaan, mutta jos toiseen luvuista on merkattu desimaalit, lisp muuntaa tuloksenkin reaaliluvuksi.

Visual Lisp Console:

```
_$ (/ 5 2)
2
_$ (/ 5 2.0)
2.5
_$ (/ 5.0 2)
2.5
```

Merkkijono (string)

Lisp ymmärtää koodin merkkijonoksi, jos sen ympäröi lainausmerkeillä. Esimerkissä alla on ensin kirjoitettu konsoliin merkkijono "tekstiä", jonka lisp ymmärtää merkkijonoksi ja palauttaa sen. Ilman lainausmerkkejä lisp ymmärtää kirjoitetun asian symboliksi, ja yrittää lukea symbolin arvoa, mutta koska sillä ei ole arvoa, lisp palauttaa nil.

Visual Lisp Console:

```
_$ "tekstiä"
"tekstiä"
_$ tekstiä
nil
```

Symbolit ja muuttujat (symbol, variable)

Symbolit on lispin tapa käsitellä muuttujia ja funktioita. Symboliin tallennetaan muuttujan tai funktion sisältö. Koodaajalle se on kuitenkin käytännössä vain muuttujan tai funktion nimi. Lispissä nimi voi sisältää kirjainten lisäksi monia merkkejä, joista yleisimmin käytössä on vain ala- ja väliviivat. Asteriskilla `*` nimen ympärillä merkataan usein, että muuttuja on yhteinen kaikille eli globaali muuttuja (malli: `*globaali-muuttuja*`), mutta tämä ei ole välttämätöntä.

Muuttujat

Jotta tietoa voidaan käyttää myöhemmin ohjelmassa, voidaan se tallentaa muistiin muuttujaan. Toisin kuin monessa muussa ohjelmointikielessä, lispissä ei tarvitse ilmoittaa muuttujan olemassaoloa ennen sen käyttöä, vaan muuttuja otetaan käyttöön automaattisesti kun se lisätään koodiin. Muuttujaan voi tallentaa mitä vain tietotyyppiä olevaa dataa. Muuttujaan tallennetaan tietoa komennolla `setq`, joka myös palauttaa muuttujaan asetetun arvon. Muuttujaa voi tallennuksen jälkeen käyttää myöhemmissä komennoissa. Muuttujan arvo ei muutu, ennen kuin siihen tallentaa uuden arvon `setq` -komennolla.

Visual Lisp Console:

```
_$ (setq numero 5)
5
_$ (+ numero 1)
6
_$ numero
5
_$ (setq tervehdys "Hei")
"Hei"
_$ tervehdys
"Hei"
_$ (strcat tervehdys " maailma!")
"Hei maailma!"
```

Funktiot

Funktioilla voidaan kirjoittaa koodia itse nimettyihin komentoihin, joita voidaan myöhemmin käyttää samalla tavalla ohjelman osana. Tämä johtaa helpommin ymmärrettävään koodiin, sillä yksittäiselle funktiolle annettu nimi kertoo paljon enemmän kuin monta riviä erillistä koodia. Funktio määritellään komennolla `defun` (lyhenne sanoista "define function").

Syötetään Visual Lisp -konsoliin alla oleva esimerkkifunktio, joka laskee kolmen luvun keskiarvon ja palauttaa reaaliluvun:

```
(defun keskiarvo (eka toka kolmas)
  (/ (+ eka toka kolmas) 3.0)
)
```

Visual Lisp Console:

```
_$ (defun keskiarvo (eka toka kolmas)
  (/ (+ eka toka kolmas) 3.0)
)
KESKIARVO
_$ (keskiarvo 3 4 5)
4.0
_$ (keskiarvo 1 4 9)
4.66667
```

Esimerkissä lisp-konsoli palauttaa funktion määrittelyn jälkeen funktion symbolin `KESKIARVO`. Tämän jälkeen keskiarvo-funktio on vapaasti käytettävissä kuten muut komennot.

Mikäli funktio halutaan käyttöön AutoCADin piirtotilassa, laitetaan funktion nimen eteen `c:` :

```
(defun c:heippa ()
  (princ "Hei maailma!")
  (princ)
)
```

Tämän jälkeen komentoa voi käyttää piirtotilassa kuten muita piirtokomentoja, kuten BLOCK tai LINE. Kun ylläolevan funktion kopioi lisp-konsoliin, piirtotilan komentoehdotuksiin ilmestyy "HEIPPA".

Funktion määrittelyn osat

Funktiomäärittelyn syntaksi koostuu nimen lisäksi neljästä osasta; nimi, argumentit, paikalliset muuttujat ja suoritettavat komennot:

```
(defun nimi (argumenttiyksi argumenttikaksi  / paikalliset-muuttujat)
  (komento ...)
  ...
  (komento ...)
)
```

Argumentit on arvoja, joita funktio käyttää suoritukseen. Nämä annetaan funktiota kutsuvasta ohjelmasta funktion parametreina. Kutsun `(nimi arvo-1 arvo-2)` kohdalla arvo-1 kopioituu argumenttiyhteen ja arvo-2 kopioituu argumenttikahteen. Tämän jälkeen funktio suorittaa komentonsa käyttäen argumenttiyhtä ja argumenttikahta.

Paikalliset muuttujat on muuttujia, joita funktiossa käytetään, mutta joiden arvon ei haluta säilyvän funktion ulkopuolella. Jos esimerkiksi asetamme muuttujaan `kokonimi` arvon `"Matti Meikäläinen"` funktion nimi sisällä, ja kerromme muuttujan olevan paikallinen, `kokonimi` unohtaa arvonsa heti kun poistutaan funktiosta.

Visual Lisp Console:

```
_$ ; Funktio globaalilla muuttujalla:
(defun nimi-globaali (etunimi sukunimi)
  (setq kokonimi (strcat etunimi " " sukunimi))
)

; Funktio paikallisella muuttujalla:
(defun nimi-paikallinen (etunimi sukunimi / kokonimi)
  (setq kokonimi (strcat etunimi " " sukunimi))
)
NIMI-GLOBAALI
NIMI-PAIKALLINEN
_$ (nimi-paikallinen "Matti" "Meikäläinen")
"Matti Meikäläinen"
_$ kokonimi
nil
_$ (nimi-globaali "Matti" "Meikäläinen")
"Matti Meikäläinen"
_$ kokonimi
"Matti Meikäläinen"
```

Lista (list)

Lista on lispin pohjimmainen tietorakenne. Lista koostuu elementeistä, jotka voivat olla mitä vain tietotyyppiä. Listojen käsittelyyn AutoLISPissä on monia valmiita funktioita. Alla on vain pintaraapaisu.

Lista voidaan luoda monella tapaa. Yksi on käyttää komentoa `list`, joka ottaa annetut arvot ja kasaa niistä listan siinä järjestyksessä kuin ne on sille annettu:

Visual Lisp Console:

```
_$ (setq listayksi (list "a" 2 "c"))
("a" 2 "c")
```

Saman voi tehdä myös antamalla listan kirjaimellisesti, jolloin kirjoitamme listan heittomerkin jälkeen `'("a" 2 "b")`. Tällöin yksittäiset elementit täytyy olla jo tiedossa, sillä heittomerkki listan edessä tarkoittaa, ettei lisp yritä suorittaa seuraavana tulevaa osaa, vaan ottaa sen sellaisenaan. Tämä tarkoittaa, ettei listan sisäisiä komentoja suoriteta, vaan ne tulkitaan listan jäseniksi.

Listaan voidaan lisätä jäseniä komennolla `cons` (tulee sanasta "construct"). Cons lisää jäsenen listan ensimmäiseksi:

```
_$ (setq listakaksi (cons "jäsen" listayksi))
("jäsen" "a" 2 "c")
```

Cons toimii joskus yllättävällä tavalla, mitä selitän assosiaatiolistojen osiossa lisää.

Listoja voidaan myös ketjuttaa toisiinsa komennolla `append`:

```
_$ (setq koottu-lista (append listayksi listakaksi))
("a" 2 "c" "jäsen" "a" 2 "c")
```

Listan jäseniin pääsee käsiksi komennoilla `nth`, `car` ja `cdr`. Komento `nth` palauttaa "ännännen" jäsenen, eli annetun järjestysnumeron mukaisen jäsenen. Järjestysnumerot lähtevät nollasta, joten jos halutaan neljäs jäsen listalta, kutsutaan `(nth 3 koottu-lista)`. `car` palauttaa listan ensimmäisen jäsenen. `cdr` palauttaa listan lopun toisesta jäsenestä lähtien. On olemassa myös komentojen `car` ja `cdr` yhdistelmiä kuten `cadr`, joka vastaa komentoa `(car (cdr lista))`. Näitä yhdistelmiä on neljään komentoon asti, eli esimerkiksi `cddddr` palauttaa listan lopun viidennestä jäsenestä lähtien. Listan voi kääntää komennolla `reverse`:

```
_$ (nth 3 koottu-lista)
"jäsen"
_$ (car koottu-lista)
"a"
_$ (cdr koottu-lista)
(2 "c" "jäsen" "a" 2 "c")
_$ (cddddr koottu-lista)
("a" 2 "c")
_$ (reverse koottu-lista)
("c" 2 "a" "jäsen" "c" 2 "a")
```

Listasta voi korvata jäsenen komennolla `subst`:

```
_$ (subst "uusi" "jäsen" koottu-lista)
("a" 2 "c" "uusi" "a" 2 "c")
_$ (subst "uusi" "a" koottu-lista)
("uusi" 2 "c" "jäsen" "uusi" 2 "c")
```

Assosiaatiolista (association list, lyhyesti assoc list)

AutoLISPin tieto on usein tallennettu assosiaatiolistoihin. Assosiaatiolista on lista, jossa kunkin jäsenen ensimmäinen osa on tunniste, ja toinen osa on sisältöä. Tämä mahdollistaa hakea listasta tietoa tunnisteen mukaan. Tätä käytetään entiteettien ominaisuuksien hallinnassa, josta kerrotaan vähän lisää myöhemmin.

Käyttötarkoituksena on tiedon jäsennys eri ryhmiin. Esimerkkinä voitaisiin luoda attribuuttilista, jossa listan jäsenen ensimmäisenä arvona on attribuutin nimi, ja sisältönä attribuutin arvo. Esimerkkinä blokki, jolla on attribuutit: Valmistaja: ABB, malli: S201-C10, nimellisjännite: 230V, nimellisvirta: 10A. Assosiaatiolistasta voi hakea tarvitun tiedon nimen perusteella komennolla `assoc`, joka palauttaa sen assiosiaation, joka vastaa annettua arvoa. Usein associa käytetään kuitenkin `cdr`-komennon kanssa, koska halutaan päästä käsiksi varsinaiseen sisältöön.

Visual Lisp Console:

```
_$ (setq johdonsuojakatkaisija '(("VALMISTAJA" . "ABB") ("MALLI" . "S201-C10")
      ("NIMELLISJÄNNITE" . "230V") ("NIMELLISVIRTA" . "10A")))
(("VALMISTAJA" . "ABB") ("MALLI" . "S201-C10") ("NIMELLISJÄNNITE" . "230V") ("NIMELLISVIRTA" . "10A"))
_$ (assoc "NIMELLISJÄNNITE" johdonsuojakatkaisija)
("NIMELLISJÄNNITE" . "230V")
_$ (cdr (assoc "MALLI" johdonsuojakatkaisija))
"S201-C10"
```

Tunnisteen ja sisällön yhdistelmä on erityinen lista. Kuten esimerkistä voi huomata, assosiaatiolistan jäsenissä on piste tunnisteen ja sisällön välillä. Tämä johtuu lispin pohjimmaisesta listarakenteesta. Listan elementti koostuu kahdesta osasta: sisällöstä ja osoittimesta listan seuraavaan osaan. Pisteellä on tarkoitus esittää näiden osien ero. Assosiaatiolistan jäsenessä `'("tunniste" . "sisältö")` toinen osa on täytetty osoittimen sijaan tiedolla. Normaali lista `'("a" "b" "c")` voitaisiin kirjoittaa oikeammin näin: `'("a" . ("b" . ("c" . nil)))`, mutta listan käyttöä on helpotettu poistamalla ylimääräiset merkit. Tämä kirjoitusmuoto toimii, jos sitä haluaa kokeilla:

```
_$ (setq oikea-lista '("a" . ("b" . ("c" . nil))))
("a" "b" "c")
```

Tästä johtuu myös `cons` komennon erikoinen toiminta yksittäisillä elementeillä:

```
_$ (cons "a" "b")
("a" . "b")
_$ (cons "a" (list "b"))
("a" "b")
_$ (cons "a" nil)
("a")
```

Kun kutsumme `(cons "a" "b")`, saamme vastaukseksi `("a" . "b")`, koska käytämme toisena osana merkkijonoa emmekä listaa. `nil` on todellisuudessa tyhjä lista `'()`, minkä takia siihen lisäämällä saa luotua yksijäsenisen listan. Seuraavassa esimerkissä näkyy todellisen listarakenteen vastaavuudet cons-komennon kanssa:

```
_$ '("a" . ("b" . ("c" . nil)))
("a" "b" "c")
_$ (cons "a" (cons "b" (cons "c" nil)))
("a" "b" "c")
_$ '("a" . "b")
("a" . "b")
_$ (cons "a" "b")
("a" . "b")
_$ '("a" . nil)
("a")
_$ (cons "a" nil)
("a")
_$ '("a" . ())
("a")
_$ (cons "a" '())
("a")
_$ '(("tunniste" . "sisältö") . (("tunniste2" . "sisältö2") . nil))
(("tunniste" . "sisältö") ("tunniste2" . "sisältö2"))
_$ (cons (cons "tunniste" "sisältö") (cons (cons "tunniste2" "sisältö2") nil))
(("tunniste" . "sisältö") ("tunniste2" . "sisältö2"))
```

Tiedosto-osoitin (file descriptor)

Tietoa voi tallentaa ja lukea tiedostosta käyttämällä tiedosto-osoittimia. Kun avaamme tiedoston lispin kautta, avoin tiedosto palauttaa muuttujaan tiedosto-osoittimen. Tiedosto-osoitin muistaa kohdan, josta tiedostoa viimeksi luettiin.

Tiedoston voi avata komennolla `open`. Komento ottaa kaksi parametria: tiedostopolun ja tiedoston avausmoodin. Tiedoston avausmoodi voi olla luku "r", kirjoitus "w" tai jatkaminen "a" (read, write, append). Lukumoodissa tiedostosta voi vain lukea tietoa, kirjoituksessa voidaan kirjoittaa tietoa olemassa olevan sisällön päälle, ja jatkamisessa kirjoitetaan tietoa olemassa olevan sisällön perään.

Tiedostosta voidaan lukea rivi kerrallaan komennolla `read-line`. `read-line` palauttaa rivin tekstinä, ja mikäli tiedosto on loppu, se palauttaa `nil`. Vastaavasti tekstiä voi kirjoittaa tiedostoon rivi kerrallaan komennolla `write-line`. Mikäli tiedostoa ei ole vielä olemassa merkityssä kansiossa ennen kirjoitusta, moodit "w" ja "a" alkavat kirjoittaa suoraan uuteen tiedostoon annetulla tiedostonimellä.

Mikäli lispille ei anna koko tiedostopolkua, se luo tiedoston tämänhetkiseen oletuskansioon.

Tiedosto täytyy muistaa sulkea käytön jälkeen, tai muut ohjelmat eivät voi muokata sitä tai tiedosto voi olla vaikea poistaa. Tiedosto suljetaan komennolla `close`. Mikäli tämä unohtuu, voi joutua käynnistämään koneen uudelleen, jotta kaikki tiedostot sulkeutuvat ja niitä voi muokata taas.

Visual Lisp Console:

```
_$ ; Avataan uusi tiedosto kirjoitettavaksi, kirjoitetaan siihen rivi ja suljetaan se.
_$ (setq uusi-tiedosto (open "testi.txt" "w"))
#<file "testi.txt">
_$ (write-line "Tämä on ensimmäinen rivi." uusi-tiedosto)
"Tämä on ensimmäinen rivi."
_$ (close uusi-tiedosto)
nil
_$ ; Avataan tiedosto jatkettavaksi, jatketaan sitä rivillä ja suljetaan se.
_$ (setq jatkettava-tiedosto (open "testi.txt" "a"))
#<file "testi.txt">
_$ (write-line "Tässäpä on toinen rivi!" jatkettava-tiedosto)
"Tässäpä on toinen rivi!"
_$ (close jatkettava-tiedosto)
nil
_$ ; Avataan tiedosto luettavaksi, luetaan se kokonaan ja suljetaan se.
_$ ; Tiedostossa on vain kaksi riviä, joten kolmas luku palauttaa nil.
_$ (setq avoin-tiedosto (open "testi.txt" "r"))
#<file "testi.txt">
_$ (read-line avoin-tiedosto)
"Tämä on ensimmäinen rivi."
_$ (read-line avoin-tiedosto)
"Tässäpä on toinen rivi!"
_$ (read-line avoin-tiedosto)
nil
_$ (close avoin-tiedosto)
nil
```

Valintajoukko (selection set)

Valintajoukot ovat kokoelma entiteettejä, eli kokoelma piirustuksen sisällöstä (entiteeteistä lisää seuraavassa osiossa). Kuvasta voi valita/hakea sisältöä annettujen tietojen perusteella. Tiedot annetaan assosiaatiolistana komennolle `ssget`. Valintajoukko tallennetaan muuttujaan aivan kuten muutkin tietotyypit `(setq valintajoukko (ssget))`. Valintajoukkoja voi olla samaan aikaan vain rajattu määrä (128), joten tallennettu valintajoukkomuuttuja kannattaa heti tyhjentää käytön jälkeen antamalla komento `(setq valintajoukko nil)`.

`ssget`:llä voi hakea entiteettejä joko tietyltä alueelta kuvasta tai koko kuvan tietokannasta. Tarkemmat tiedot löytyy AutoCADin lisp-referenssistä, johon pääsee CADin helpin kautta. Eniten kuvan käsittelyssä tulee kuitenkin käytettyä kolmea moodia: koko kuvan tietokannasta hakua, valmiista valinnasta hakua tai annetaan käyttäjän valita. Jos ei anneta käyttäjän valita, `ssget` ottaa kaksi parametria: valintamoodin ja assosiaatiolistan haettavista ominaisuukista. `"X"` tarkoittaa, että haetaan asioita koko kuvan tietokannasta, ja `"I"` tarkoittaa, että haetaan sopivia entiteettejä käyttäjän valmiiksi valitsemalta alueelta. Kun moodi-parametri jätetään pois, Lisp odottaa, että käyttäjä valitsee kuvasta haluamansa asiat. Jos esimerkiksi halutaan valita kaikki text-tyyppiset entiteetit kuvasta, käytetään komentoa `(ssget "X" '((0 . "TEXT")))`. Mikäli haku halutaan rajata valmiiksi valittuihin asioihin, vaihdetaan X:n paikalle I: `(ssget "I" '((0 . "TEXT")))`. Ilman moodia käyttäjä saa valita tekstit itse, ja valinta ei ota muita entiteettejä ollenkaan huomioon `(ssget '((0 . "TEXT")))`. Jos halutaan antaa käyttäjälle täysi vapaus valintaan, ei anneta mitään parametreja: `(ssget)`.

`sslength` palauttaa valintajoukon pituuden, eli sen, montako entiteettiä on valittuna. `ssname` palauttaa annetun järjestysnumeron mukaisen entiteettinimen. Jälleen, järjestysnumerot lähtevät nollasta. Esimerkissä avoinna on piirustus, jossa on yksi teksti ja 72 blokkia. 0 on assosiaatiotunniste, joka viittaa entiteettityyppiin. Näistä arvoista lisää seuraavassa kappaleessa, ja tiedostossa DXF-koodeja.md.

Visual Lisp Console:

```
_$ (setq tekstit (ssget "X" '((0 . "TEXT"))))
<Selection set: 71>
_$ (sslength tekstit)
1
_$ (ssname tekstit 0)
<Entity name: 1e57bdcdc30>
_$ (setq tekstit nil)
nil
_$ (setq blokit (ssget "X" '((0 . "INSERT"))))
<Selection set: 73>
_$ (sslength blokit)
72
_$ (ssname blokit 20)
<Entity name: 1e57bdd0910>
_$ (setq blokit nil)
nil
```

Entiteettinimi / kokonaisuustunniste (entity name)

Entiteetit ovat AutoCAD-piirustuksen sisältöä, esimerkiksi blokkeja, viivoja, tekstejä tai attribuutteja. Kullakin entiteetillä on oma tunniste, jolla lisp pääsee sitä käsittelemään. Tätä tunnistetta kutsutaan entiteettinimeksi. Kullakin entiteetillä on kokoelma ominaisuuksia, joita pääsee lispistä käsin muokkaamaan. Ominaisuudet esitetään assosiaatiolistassa, jossa kutakin ominaisuutta vastaa jokin numero. Näitä kutsutaan DXF-koodeksi*.

Ominaisuuden muokkaus voidaan toteuttaa komennoin:

- valitaan kuvasta halutut asiat: `ssget`
- selvitetään muokattavan asian entiteettinimi: `ssname`
- haetaan tämän entiteetin ominaisuuslista: `entget`
- vaihdetaan listasta haluttu ominaisuus: `subst`
- tallennetaan muutettu lista entiteettiin: `entmod`
- päivitetään entiteetti: `entupd`
- tyhjennetään valintajoukko

Assosiaatiolistat ovat melko pitkiä, joten niitä on lyhennetty tässä korvaamalla osia kolmella pisteellä "...".

Visual Lisp Console:

```
_$ (setq kaikki-tekstit (ssget "X" '((0 . "TEXT"))))
<Selection set: 6>
_$ (setq tekstientity (ssname kaikki-tekstit 0))
<Entity name: 23d99607c30>
_$ (setq entOminaisuudet (entget tekstientity))
((-1 . <Entity name: 23d99607c30>) (0 . "TEXT") ... (1 . "LAITE-ERITTELY") ...)
_$ (setq entOminaisuudet (subst '(1 . "uusi teksti") (assoc 1 entOminaisuudet) entOminaisuudet))
((-1 . <Entity name: 23d99607c30>) (0 . "TEXT") ... (1 . "uusi teksti") ...)
_$ (entmod entOminaisuudet)
((-1 . <Entity name: 23d99607c30>) (0 . "TEXT") ... (1 . "uusi teksti") ...)
_$ (entupd tekstientity)
<Entity name: 23d99607c30>
_$ (setq kaikki-tekstit nil)
nil
```

Attribuutit ovat erityisiä siinä, ettei niitä pysty suoraan valitsemaan, vaan niihin pääsee käsiksi vain blokin kautta. Eli ensin valitaan blokki, selvitetään sen entiteettinimi, jonka jälkeen aletaan kulkea blokin alaisia entiteettejä `entnext`-komennolla. `entnext` palauttaa seuraavan entitynimen tietokannassa, ja jos blokilla on attribuutteja, nämä ovat listana blokin jälkeen. Attribuutit on käyty läpi, kun seuraava entiteettityyppi (DXF-koodi 0) ei enää ole "ATTRIB". Esimerkissä valitaan blokki nimeltä "BJR_TUNNUS" ja käydään sen attribuutit läpi. Blokilla on kuusi attribuuttia: PROSESSIASEMA, JR_PA, NAYTTO, JR_NAUTTO, POS ja JR_HUONE. Attribuutin näkyvä tekstiarvo on DXF-koodi 1. Blokin tai attribuutin nimi on DXF-koodi 2.

Visual Lisp Console:

```
_$ (setq bjr-tunnus-valinta (ssget "X" '((0 . "INSERT") (2 . "BJR_TUNNUS"))))
<Selection set: 64>
_$ (setq bjr-tunnus-blokki (ssname bjr-tunnus-valinta 0))
<Entity name: 23d99601c50>
_$ (entget bjr-tunnus-blokki)
((-1 . <Entity name: 23d99601c50>) (0 . "INSERT") ... (66 . 1) (2 . "BJR_TUNNUS") ...)
_$ (setq alientity (entnext bjr-tunnus-blokki))
<Entity name: 23d99601c60>
_$ (entget alientity)
((-1 . <Entity name: 23d99601c60>) (0 . "ATTRIB") ... (1 . "PROSESSIASEMA") ... (2 . "PROSESSIASEMA") ...)
_$ (setq alientity (entnext alientity))(entget alientity)
<Entity name: 23d99601c70>
((-1 . <Entity name: 23d99601c70>) (0 . "ATTRIB") ... (1 . "FP01") ... (2 . "JR_PA") ...)
_$ (setq alientity (entnext alientity))(entget alientity)
<Entity name: 23d99601c80>
((-1 . <Entity name: 23d99601c80>) (0 . "ATTRIB") ... (1 . "NÄYTTÖ") ... (2 . "NAYTTO") ...)
_$ (setq alientity (entnext alientity))(entget alientity)
<Entity name: 23d99601c90>
((-1 . <Entity name: 23d99601c90>) (0 . "ATTRIB") ... (1 . "x.x.x") ... (2 . "JR_NAUTTO") ...)
_$ (setq alientity (entnext alientity))(entget alientity)
<Entity name: 23d99601ca0>
((-1 . <Entity name: 23d99601ca0>) (0 . "ATTRIB") ... (1 . "") ... (2 . "POS") ...)
_$ (setq alientity (entnext alientity))(entget alientity)
<Entity name: 23d99601cb0>
((-1 . <Entity name: 23d99601cb0>) (0 . "ATTRIB") ... (1 . "") ... (2 . "JR_HUONE") ...)
_$ (setq alientity (entnext alientity))(entget alientity)
<Entity name: 23d99601cc0>
((-1 . <Entity name: 23d99601cc0>) (0 . "SEQEND") ...)
_$ (setq bjr-tunnus-valinta nil)
nil
```

*) AutoDeskin kehittämä piirustustallennusmuoto on DXF, "Drawing eXchange Format". Tämän formaatin tarkoituksena oli tehdä AutoCAD-kuvista jaettavia eri suunnitteluohjelmistojen kesken. Dwg-formaatti on kompressoitua DXF-tietoa. AutoDesk edelleen hallinnoi DXF-formaattia. DXF-koodeista saa lisää tietoa hakemalla AutoCADin dokumentaatiosta "DXF". Tiedostossa DXF-koodeja.md on kuitenkin muutama tärkeimmistä arvoista.

VLA-objekti (VLA-object)

Kuvan sisältöä voi muokata entiteetti-esitystavan lisäksi myös VLA-objekteina. Tämä toiminnallisuus on saatavilla vain Windowsilla.

VLA-objektit kuuluvat AutoCADin ActiveX-objektirakenteeseen, ja niitä muokataan Visual Basic for Applications (VBA) tyyppisillä komennoilla. ActiveX:n / Common Object Modelin kautta pääsee käsiksi myös muihin ohjelmiin (kuten Exceliin). AutoCADissä tuli ennen VBA-kehitysympäristö mukana, mutta tätä kieltä ei tueta enää Microsoftin kautta, joten se pudotettiin AutoCADin oletussisällöstä. Tämän saa kuitenkin vielä ladattua laajennoksena AutoCADin sivuilta. AutoCADin ActiveX-rakennetta kuitenkin tuetaan Visual Lispin kautta. Tämä antaa pääsyn VBA:n komentoihin vla- ja vlax-laajennoksilla. Laajennos otetaan käyttöön antamalla komento `(vl-load-com)`. Tämän mukana tulee monia muitakin apufunktioita kuin ActiveX-rakenteeseen pääsevät komennot.

Lisp entiteettinimen pystyy muuttamaan VLA-objektiksi komennolla `vlax-ename->vla-object` ja takaisin `vlax-vla-object->ename`. VLA-objektin ominaisuuksia (property) voi hakea komennolla `vlax-get-property` ja muuttaa komennolla `vlax-put-property`. Objektin kaikki ominaisuudet saa "dumpattua" komennolla `vlax-dump-object`. VLA-objektikomennot tekevät ominaisuuksien muokkamisesta helpompaa, sillä ominaisuuksien nimet ovat selkokielisiä DXF-koodien sijaan.

Visual Lisp Console:

```
_$ (vl-load-com)
_$ (setq ss (ssget "X" '((0 . "TEXT"))))
<Selection set: 71>
_$ (setq tekstiobjekti (vlax-ename->vla-object (ssname ss 0)))
#<VLA-OBJECT IAcadText 0000023d9a5ce468>
_$ (vlax-dump-object tekstiobjekti)
; IAcadText: AutoCAD Text Interface
; Property values:
;   Alignment = 0
;   Application (RO) = #<VLA-OBJECT IAcadApplication 00007ff7142c70d8>
;   Backward = 0
;   Document (RO) = #<VLA-OBJECT IAcadDocument 0000023d995a2008>
;   EntityTransparency = "ByLayer"
;   Handle (RO) = "742B"
;   HasExtensionDictionary (RO) = 0
;   Height = 5.0
;   Hyperlinks (RO) = #<VLA-OBJECT IAcadHyperlinks 0000023d9f4052c8>
;   InsertionPoint = (331.294 279.259 0.0)
;   Layer = "0"
;   Linetype = "ByLayer"
;   LinetypeScale = 1.0
;   Lineweight = -1
;   Material = "ByLayer"
;   Normal = (0.0 0.0 1.0)
;   ObjectID (RO) = 152
;   ObjectName (RO) = "AcDbText"
;   ObliqueAngle = 0.0
;   OwnerID (RO) = 43
;   PlotStyleName = "ByLayer"
;   Rotation = 0.0
;   ScaleFactor = 1.0
;   StyleName = "ISO Proportional"
;   TextAlignmentPoint = (0.0 0.0 0.0)
;   TextGenerationFlag = 0
;   TextString = "LAITE-ERITTELY"
;   Thickness = 0.0
;   TrueColor = #<VLA-OBJECT IAcadAcCmColor 0000023d9f405680>
;   UpsideDown = 0
;   Visible = -1
T
_$ (vlax-get-property tekstiobjekti "TextString")
"LAITE-ERITTELY"
_$ (vlax-put-property tekstiobjekti "TextString" "uusi teksti")
nil
_$ (vlax-get-property tekstiobjekti "TextString")
"uusi teksti"
_$
```

Lähteet

AutoCADin sivustolta 15.10.2019

- About Data Types (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-7E568541-F1D0-49C4-B878-15880486825F-htm.html
  - About Integers (AutpLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-EF6114FC-F1E4-4C71-91CC-07D01E6C8ABB-htm.html
  - About Reals (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-F2BFD097-295B-4499-BBD9-0A785C377070-htm.html
  - About Strings (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-D7C33A49-9715-4E9B-9D8A-4C2E7BFC0FE5-htm.html
  - About Lists (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-F8074AA9-F361-4960-B628-681465B74229-htm.html
  - About Selection Sets (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-E0CCD0D3-AB95-40CE-A5DD-8E9214DC5469-htm.html
  - About Entity Names (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-9CB07C25-4439-445D-B1B3-92174C53C571-htm.html
  - About VLA-objects (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-F6477DD7-7982-4742-95CA-F30BBBBE80B1-htm.html
  - About File Descriptors (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-EDADF856-4D64-4E19-A7BD-5017C4135A01-htm.html
  - About Symbols and Variables (AutoLISP) https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2018/ENU/AutoCAD-AutoLISP/files/GUID-1855E7B0-2474-49E6-9EE3-8BC541FC1CC8-htm.html