

X10 Verzió 4.9

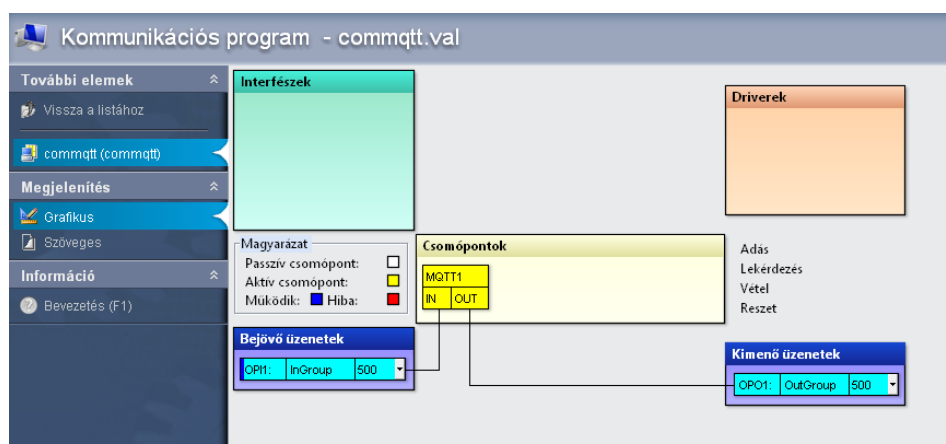
1. MQTT kommunikáció

Önálló driverként érhető el a 4.9-es verziótól kezdve az MQTT szolgáltatás. A program működtetéséhez szükség van a VISION-ön kívül még egy további komponensre, az ún. MQTT brókerre. A bróker ennek a kommunikációnak a közös névtére, valójában a pillanatnyi adatokat tároló szerver. Jelenleg a VISION csak a Mosquitto brókert tudja használni, azzal teszteltük. Elérhető minden létező platformra és a felhőszolgáltatók is támogatják. Az IoT eszközök körében elterjedt MQTT kommunikációról azt kell tudni, hogy ebben a rendszerben nem egymással közvetlenül, hanem a brókerrel, ill. a brókeren keresztül kommunikálnak az eszközök. Így a VISION is. Amikor egy eszköz a brókerhez kapcsolódik, feljelentkezik rá adott felhasználó névvel, jelszóval és a szolgáltatási szint megadásával, majd feladhat adatokat a szerverbe. Az adatokat az MQTT-ben topikokba szervezik; egy-egy topik bármilyen struktúrájú és adatszerkezetű lehet, de a konvenciókat követve a kódolást többnyire JSON-ban követik el. Így a VISION is ehhez a JSON konvencióhoz ragaszkodik: a konkrét változókat a JSON kódszerkezet dekódolásával állapítja meg és rendeli a topikon belül önálló változókhoz. Mindjárt megmutatom, hogyan.

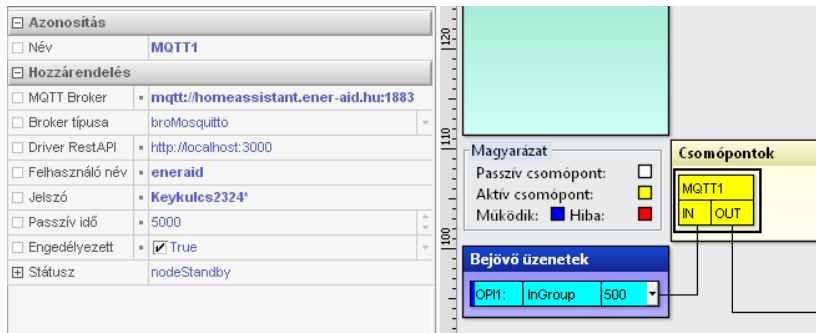
De előtte még a kommunikáció módját kell megérteni. A kommunikáció végpontjai mind kliensei a brókernek, ahonnan lekérdezhető többek között az összes topik. Majd feliratkozhat a kliens adott topikokra vagy az összes topikra egyetlen utasítással. A VISION-ben automatikusan ez utóbbit csináljuk: a brókerhez való kapcsolódás után tehát minden adat elérhető lesz a VISION számára, ami ugyanazzal a felhasználó névvel és jelszóval lett regisztrálva. A bróker egyszerűen azt csinálja, hogy amennyiben adatot kap az egyik csatornán, azt elküldi az adott topikra feljelenetkező összes többi kliensnek. Gondoljunk egy hőmérőre. Amikor a hőmérséklet megváltozik, az új adatot az eszköz felküldi a brókernek. Az meg tovább a VISION-nek. És fordítva is ez történik: amint a VISION adatot ír a brókerbe, az továbbításra kerül a bróker által az összes kliens felé, akiket a kérdéses topik érdekel. Így lesz végül változásfüggő a kommunikáció a kliensek között, ill. a VISION és a szenzorok között.

MQTT konfigurációja

A rendszer beállítása szinte teljesen azonos azzal, amit az OPC-UA esetén megismertünk – csak a fogalmak és az elnevezések különbözőek:



Pont úgy kell feltenni és beállítani, mint az OPC-UA-t. Csak itt nem az eszközhöz közvetlenül, hanem a brókerhez kapcsolódunk:

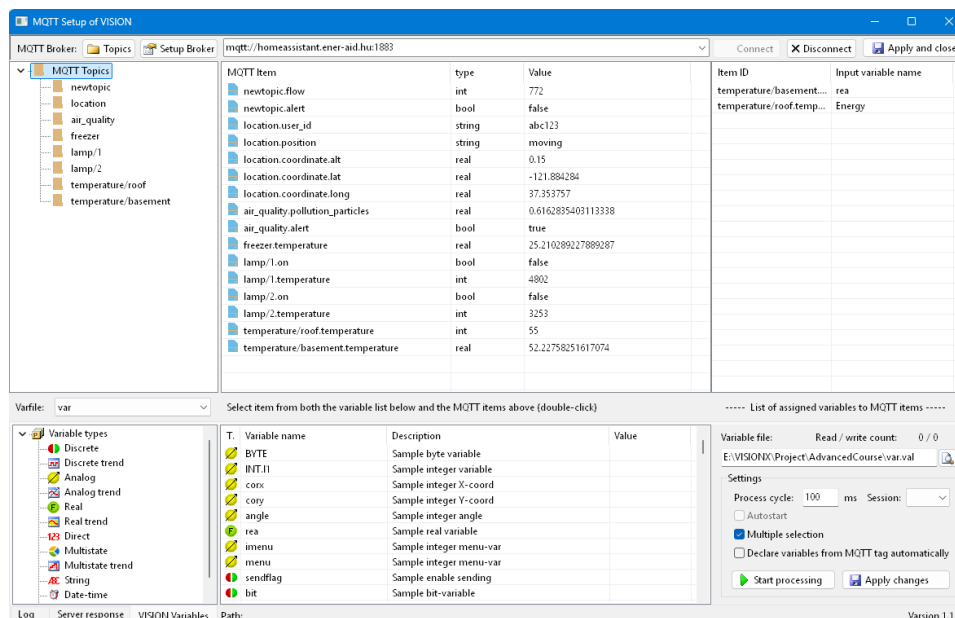


A szolgáltatás azonosítása az mqtt:// előtaggal történik, utána IP cím és portcím, a szokásos módon.

A képen egy homeassistant-ban installált brókerhez való kapcsolódást láthatunk. Ott van hozzá a tesztekhez a

felhasználó név és a jelszó is. Magával a driverrel RestAPI-t használva kommunikál a program, ami mindig a kommunikációt folytató gépen indul el, automatikusan, a 3000-es porton. Ezért lehet hivatkozni rá localhost:3000-el.

Az adatok beállítása az OPC-UA mintáját követi:



Jobbra, a fastruktúrában a topikok látszanak, rákattintva a topik-struktúra elemeire pedig láthatóvá válik az összes JSON-ból dekódolt MQTT item középen. Ezeket húzhatjuk a jobb felső összerendelő táblázatba, alulról pedig VISION változót rendelhetünk hozzá drag-&-droppal, vagy dupla kattintással – a szokásos módon.

A kommunikációt ugyanúgy tesztelhetjük a *Start processing* gombbal, az eredmények a Log fülön látsznak. Azok számára, akiket a JSON kódszerkezet is érdekel, a *Server response* fület javasolom megtekinteni.

A topiklista elején látható **newtopic** ebben a példában a VISION által létrehozott elem, deklarációja ennyi:

Bármilyen nevet megadhatunk, a pont előtti tag lesz az általunk generált topik neve. Így valósul hát meg a bevezetőben említett működési mód.

```
OP01:OPCO
BEGIN
  NodeRef=      @MQTT1
  Data=         "newtopic.flow", Water,
               "newtopic.alert", bit
END
```

X10 Verzió 4.8

1. Kamera streaming Home Assistantból

A Home Assistant integrációjáról [ebben a fejezetben](#) írtunk részletesen. A rendszeren belül a kamerák kezelésére készült egy egyszerűbb eljárás. Ennek az a lényege, hogy mostantól elég a *Streaming* entitás nevét megadni az LD egy új tulajdonságánál. Ugyanitt megadható a frissítés ciklusideje is (*StreamRefresh*). Abban az esetben, ha a kamera nem elérhető, az LD-nek átadott eredeti képet jeleníti meg a program (*Képnév*). Ez előnézetként használható, ami a kamera által megmutatott képre utal (pl. az épület előteréről készült screenshot):

```

20 LD1: LD "ProjectBanner.bmp", bm_Trans
21 BEGIN
22     Position=      24.75, 116.625
23     Streaming=     HA.camera.camera1_sub
24     StreamRefresh= 5000
25 END

```

Hozzárendelés	
☐ Képnév	ProjectBanner.bmp
☐ Változó	☒ True
☐ Default kép	
☐ Streaming	HA.camera.camera1_
☐ Stream Refresh	5000

Az új tulajdonság a *Streaming*, ami egy *camera* változót igényel (domain név a HA-ban). Alatta látható a *StreamRefresh*, ami miliszekundumban értendő. A hozzárendelés tetején olvasható *Képnév* az előnézeti kép neve.

2. Profilkezelés Fact-IoT-ban

A Fact-IoT rendszer konfigurációja mostanra eléggé bonyolult lett – több fülön kell odafigyelni a beállításokra. A konfiguráció névvel azonosított eltárolására szolgál a profil, ami a Setup fülön található (jobb oldalt):

ID	Description
50252	Kórházi szerver konfiguráció
62691	Bácsvíz szuperprojekt
21226	Automatikus akármilyen projekt

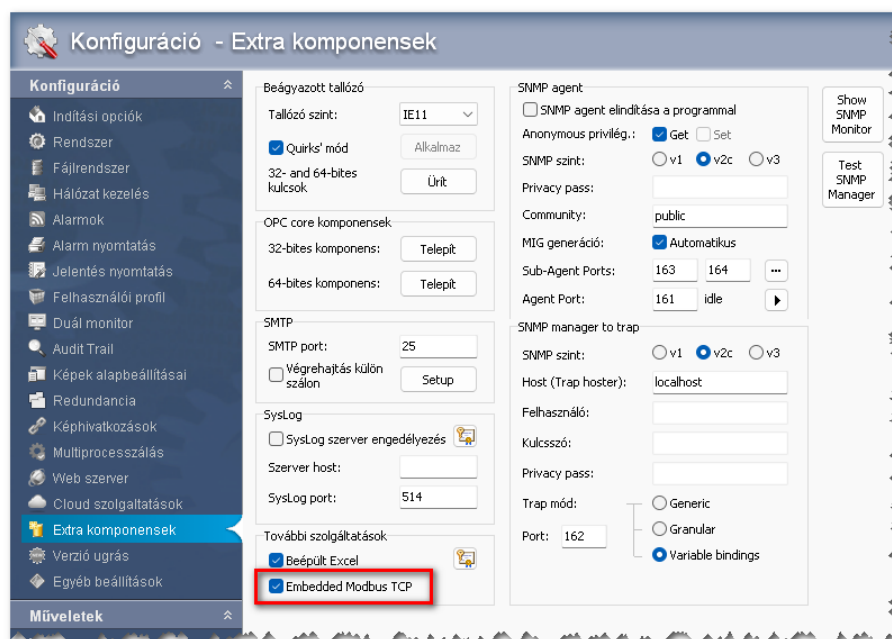
A lényeg, hogy a beállításhoz a rendszer automatikusan egy egyedi azonosítót rendel és a FactIoT.ini valamint FactIoTPrj.csv fájlokat ez alatt tárolja a Profiles alkönyvtárban belül. Egy-egy régebbi beállítás visszatöltése egyszerűen a Profil-lista megfelelő sorára való dupla kattintással történik. Ezután a szervert újra kell indítani, hogy az új beállítások érvényre jussanak. Ne felejtjük el, hogy futó alkalmazások lehetnek a háttérben, amit le kell zárni és újat indítani – ezért szükséges újraindítás.

X10 Verzió 4.7

1. Bépülő ModTCP kommunikációs driver

A kommunikációs driverek száma sajnos még a 64-bites rendszerben is korlátozott. Ennek a Microsoft Windows-ra visszavezethető okai vannak. A párhuzamosan futó driver-szám 100 körüli. Szerverek esetén valamivel több, Windows 10/11 gépeken alacsonyabb. Részben ennek a problémának a kezelésére fejlesztettük ki a MODTCPX drivert, amiről többet [itt olvashattok](#). Megjegyzem, más rendszerekben is vannak hasonló korlátok. Bosszantó, ismert probléma tehát.

Megoldására készült a beépülő Modbus TCP opció, aminek a beállítása látható az alábbi konfigurációs képen az *Extra komponensek* belül:



Amennyiben bekapcsoljuk, a program nem tölt be külön drivert, a ModTCP kommunikációt önmagán belül létrehozott komponenssel kezeli. Ezek száma nem korlátozott. Egyszerűbb és kényelmesebb is, mint a külön driver használata, ráadásul valamivel még gyorsabb is.

Miért épp a ModTCP-t tettük beépíthetővé? Megfigyelésünk szerint ez a leggyakoribb kommunikáció jelenleg az alkalmazásokban, a száma könnyen többszáz fölé nőhet. Ezért indokolt volt erre a gyakori driverre fókuszálni, annak a betöltését és futtatását gyorsítani és vele a betöltött driver-számot jelentősen csökkenteni.

A használata ingyenes, senkit ne tévesszen meg a *Beépült Excel* checkbox mellett található licence bekérő gomb. Az az Excel-re vonatkozik.

Érdemes tudni még, hogy a **Fact-IoT használata esetén a probléma nem létezik**, ott nem találkoztunk még hasonló korlátozó tényezőkkel.

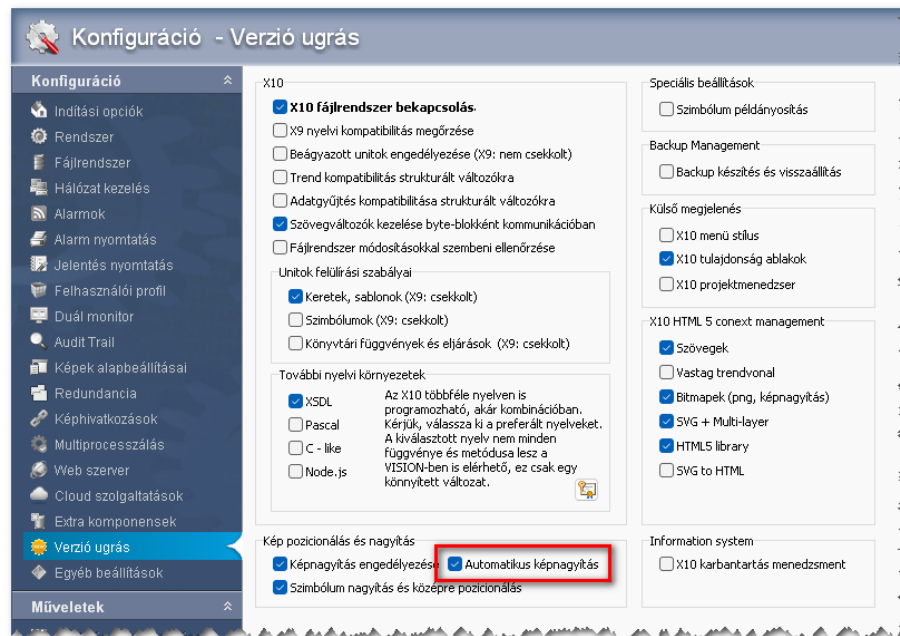
2. VPR.INI speciális paraméterek

A használat során számos speciális paraméter került bevezetésre a VPR.INI fájlban, de nem mind lett rávezetve a már amúgy is komplikált konfigurációs oldalakra. Elég speciális magyarázat is fűződik mindegyikhez, és mind a praktimuot szolgálja. Ezek felsorolása és magyarázata következik alább:

System parameter (VPR.INI)	Funkciója, jelentése
LogSystemErrors	Előfordulhatnak rendszer szintű hibák, amiket a rendszer lekezel. Az opció azt állítja be, hogy ezeket a rendszer naplózza-e. Default = 1
OnErrorRestart	Ha 1 az értéke, akkor ugyanannak a rendszer szintű hibának a bekövetkezése esetén, amennyiben arra 30 másodpercen belül került sor, a program újraindul (nem javasolt; default=0)
FactIoTprefix	Ezzel állítható be, hogy egyazon gépen belül több, különböző IloT szervert indíthassunk el - pl. distribution szerver; szervert és klienset stb. A paraméter az osztott memória prefix-azonosítójaként szolgál.
SQLTrendSaveOnExit	Beállítása (1) azt eredményezi, hogy kilépéskor a rendszer a trendek DATATR.N cache fájlban tárolt tartalmát az SQL szerverbe is beírja. Ez egyrészt nagy rendszerek esetén eléggé lassú, másrészt nem feltétlenül szükséges, ha még aznap visszaindítjuk a szervert. Hiszen az összes adat ott van fájlként a cache-ben. Kikapcsolásával jelentősen felgyorsul a kilépés.
FlushTimeDivider	A mintavételezés nem azonnal menti az adatokat az SQL adatbázisba, mert az reménytelenül lassú lenne. Ehelyett cache-eli azokat és egy független flush ciklus végzi a periodikus mentést. Ennek gyakoriságát a program automatikusan számítja ki, de szükség esetén felgyorsítható. Akár a 10-szeresére is, ha ebbe a paraméterbe 10-et írunk.
FlushTrendChecker	Kiterjedt, több szerver-kliens hálózati csomópontból álló elosztott rendszernél használt annak felgyorsítására, hogy a kliensek által épp vizsgált trendek minél gyorsabban frissüljenek. Megj.: IloT használata esetén ez szükségtelen, ott a trend mindig realtime.
DisableDummies	A VISION az adattáblák végéhez automatikusan hozzáfűz 3 plusz változót annak érdekében, hogy a változót igénylő objektumok (BV, NU, TD pl.) tényleges változóhozzárendelés nélkül is felrajzolhatók legyenek. A változólista elvetése esetén ui. a program a Dummy-t használja és ezért nem jelez hibát. Ha azonban ezt letiltjuk, a változó hozzárendelése kötelező lesz (előbb-utóbb amúgy is). A kikapcsolás mellett szól továbbá, hogy sok kis tábla esetén így feleslegesen sok plusz változó keletkezne - feleslegesen.
NoMenuSelectWhenCtrl	Abban az esetben, ha az SQ menüt használjuk egy képrészlet kiválasztására és hozzá popup ablak megjelenítésére, akkor az SQ-n belüli objektumok kiválaszthatatlanná válnak! Pedig pl. hasznos lenne a kiválasztásuk pl. a TrendWindow miatt. Ilyenkor az a trükk, hogy a Ctrl-t nyomva kell tartani és úgy kell rákattintani az objektumra. Ez viszont csak akkor lehetséges, ha ez az opció aktív (=1).
EmbeddedModtcp	Beépülő Modbus TCP driver a DLL korlát kiküszöbölése érdekében

3. Automatikus képnagyítás

Általában nem javasolt a folyamatképek, pláne az összetett folyamatképek projekt-felbontásától eltérő megjelenítése, mégis szükség lehet rá abban az esetben, ha egy alacsony felbontású projektet modernizálunk és ennek során nagyobb felbontású monitoron szeretnénk azt megjeleníteni. Erre szolgál az verzió ugrás konfigurációs lapon látható alábbi opció:



Különösen látványos a vektorgrafikus nagyítás, amennyiben Full-HD képet szeretnénk 4K felbontású monitorra tenni. Az 2-szeres nagyítás mindig tökéletes.

Az automatikus képnagyítást természetesen a *képnagyítás engedélyezésével* együtt kell használni. Érdemes továbbá kipróbálni a *szimbólum nagyítás és középre igazítás* opciót is, ami a szimbólumok szerkesztését segíti, kiemelten az SVG képes objektumokra.

4. Adatok szinkronizációja

Készült egy eljárás, ami bármilyen projekt bármilyen változó-részhalmazának, unitjának, vagy több unitjának az átvitelét teszi lehetővé egy másik projektbe, vagy ugyanazon projekten belül egy másik unit-ba. De használható ugyanazon a projekten belül is. Az új parancs szintaktikája a következő:

SynchronizeData (forrás, cél, opcionális kivétel)

Például:

SynchronizeData ("@PRJ1.UN1.*", @PRJ2.MASIKUNI.*")

Prototípus deklarációja:

```
SynchronizeData::      procedure (source: string; destination: string; exception: optional string)
                        'Exchange changes (synchronize) between two variable sets identified by wildchars like U1.*, U2.*;
                        'The source is the dominant and used during initialization
```

Az átszinkronizálás változás alapú átvitel takar. Ha kiadunk pl. egy parancsot az egyik másolatban, megjelenik a másikban és megfordítva. Nagyon hasznos különböző félépítésű változók között.

X10 Verzió 4.6

1. DFS helyettesítő hálózati kommunikáció a Fact-IoT rendszerben

A Fact-IoT – *distribution service* alapú hálózati kapcsolatok megkönnyítésére készült egy új megoldás a Fact-IoT szerverben. A változtatás célja, hogy kiterjedt (adott esetben többszáz végpontot tartalmazó) rendszerek konfigurálása a központból elvégezhető legyen és ne kelljen önálló beállításokat elkövetni a kliensek oldalán, ahol a *distribution service* fut.

Emiatt hoztuk létre a *Remote projects* oldalt a Fact-IoT szerver konfigurációs programon belül:

Factory IoT System of VISION

Welcome Local and shared projects Networking Settings Remote projects Integration Admin 859 ; Trend Seek: 30.011 ms - 102

Remote project data

Computer name: Project name:

IP address of server: Remote VPR file:

Workstation identifies itself by checking the above computer name which must therefore be defined. The IP address (host name) however may be different from the client's point of view.

Direction of sync: ☐ Remote to server ☒ Server to remote comp.

Frequency of sync: ☐ Every minute ☒ Every hour ☐ Midnight

Scope of sync: ☐ Var only ☒ Application ☐ Whole system

Activate association: ☒ Enabled

New remote project Delete remote project Save modifications Package-based synchronization

Enabled	Computer name	Server's IP address	Remote project name	Remote VPR file	Direction	Sync frequency	Scope of sync
-	beast		Manning	E:\VISIONX\Project\Elcon\Manning\Man...	Server to remote	Hour	Application
ok	beast		Peterfy_Peterfy	E:\VISIONX\Project\Elcon\Peterfy_Peterfy...	Server to remote	Hour	Application
-	beast		Denso	E:\VISIONX\Project\Elcon\Denso\Denso.vpr	Server to remote	Hour	Application

V1.5 ini: Default Copyright © PROVICON 1994-2025

Itt tetszőleges számú végpontot definiálhatunk, amelyben a lokális gépen futó alkalmazást, alkalmazásokat soroljuk fel. Egyetlen ilyen lista készül a központi szerveren, és minden végpont ugyanazt a listát látja. **A lokális gép a *Computer name* oszlop alapján ismeri fel önmagát.** Amennyiben a gépet például BEAST-nek hívják (ld. a listát), akkor a kliens automatikusan *distribution service*-ként indul el és betölti a megadott projektet. Illetve projekteket. Mivel a listában, ugyanahhoz a géphez rendelve több projekt is létrehozható – ezzel lokálisan egy superprojektet hozva létre. A lényeg, hogy a szerveren és a kliensen ugyanaz a konfiguráció fut (valójában szerver beállítással).

Sokkal kényelmesebb és egyszerűbb módja ez a több tucat alkalmazást tartalmazó superprojektek kezelésének, és elosztott rendszerek létrehozásának, ahol a végpontok a Data Lake projektjeinek bármilyen részhalmozát képezik úgy, hogy közben minden adat rendelkezésre áll mindenhol – az IIoT filozófiájának megfelelően. Jogosultságtól függ csak, hogy ki, mihez férhet hozzá, a fenti listától pedig az, hogy a lokalitások mit futtatnak.

A *distribution service*-ként való futás ténye a *Networking* fülön figyelhető meg:

Factory IoT System of VISION

Welcome Local and shared projects Networking Settings Remote projects Integration Admin 321 - 102

Settings ☒ Autostart ☐ Redundant system ☒ Requests ☐ Headers ☐ Web log Quick cycle (ms): 10 Cycle time: 100

☐ Client (DS) Port: 8090 Get master

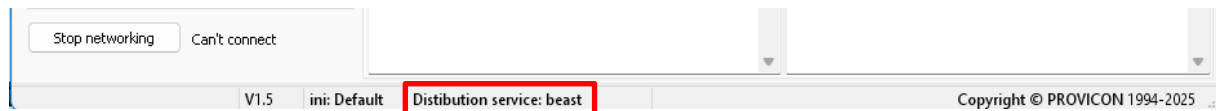
☒ Server Host: beast:8081

☒ Distribution service

☐ Automatic server-client select by path (\\)

Sessions Requests Headers Statistics

Valamint a státuszszorban:



Konfiguráció Excel-ben

A fent vázolt lista előállítható Excel segítségével is, lévén ez egy szövegfájl *FactIoTPrj.csv* néven a *Bin* könyvtárban. Így tényleg könnyen előállítható és editálható akár több 100 végpont is:

File	Home	Insert	Page Layout	Formulas	Data	Review	View	Automate	Help	Acrobat
Clipboard: Paste, Copy, Cut, Format Painter Font: Aptos Narrow, 11, Bold, Italic, Underline, Color, Background Color Alignment: Left, Center, Right, Justify, Merge & Center, Wrap Text Number: General, Percentage, Decimals, Fractions, Text, Date, Time Styles: Conditional Formatting, Format as Table, Normal, Good										
A10										
	A	B	C	D	E	F	G	H		
1	Enable	RemoteComputer	ServerIP	RemoteProjectName	VPRProjectFile	SyncDirection	SyncFrequency	SyncScope		
2	-	beast		Manning	E:\VISIONX\Project\Elcon\Manning\Manning.vpr	Server to remote	Hour	Application		
3	ok	beast		Peterfy_Peterfy	E:\VISIONX\Project\Elcon\Peterfy_Peterfy\BP_Peterfy_Peterfy_PSPS.vpr	Server to remote	Hour	Application		
4	-	beast		Denso	E:\VISIONX\Project\Elcon\Denso\Denso.vpr	Server to remote	Hour	Application		
5										

A *SzerverIP* (IP address of server) oszlopról annyit kell tudni, hogy abban az esetben szükséges, ha a lokális hálózati csomópont szempontjából más IP címen látszik a szerver, mint a szerver lokális hálózatában. Ami könnyen elképzelhető egy router mögött, ill. VPN hálózaton.

2. Hálózati üzemmód detektálásának egyszerűsítése

A Fact-IoT rendszer bevezetésével a hálózati csomópont már egyszerűsíthető szerverre és kliensre és nem csekkelhető csupán a *MasterVision* és a *NetServer* függvényeket használva.

Ha ehhez hozzávesszük még a redundáns rendszerek lehetséges állapotait, könnyen belátható, hogy a kombinációk száma alaposan megnőtt. Például a standby szerver azonosítható a *StandbyServer* nevű új függvénnyel, az pedig, hogy a hálózat egyáltalán működik-e, az *AnyNet* függvénnyel. Készült ezért még egy további függvény, ami mindezt egyetlen string-függvényben egyesíti:

NetNodeState:: shortstring function

'get the state of the active client workstation as string (Net server, Standby server, Server, Client)

Lehetséges állapotai a következő:

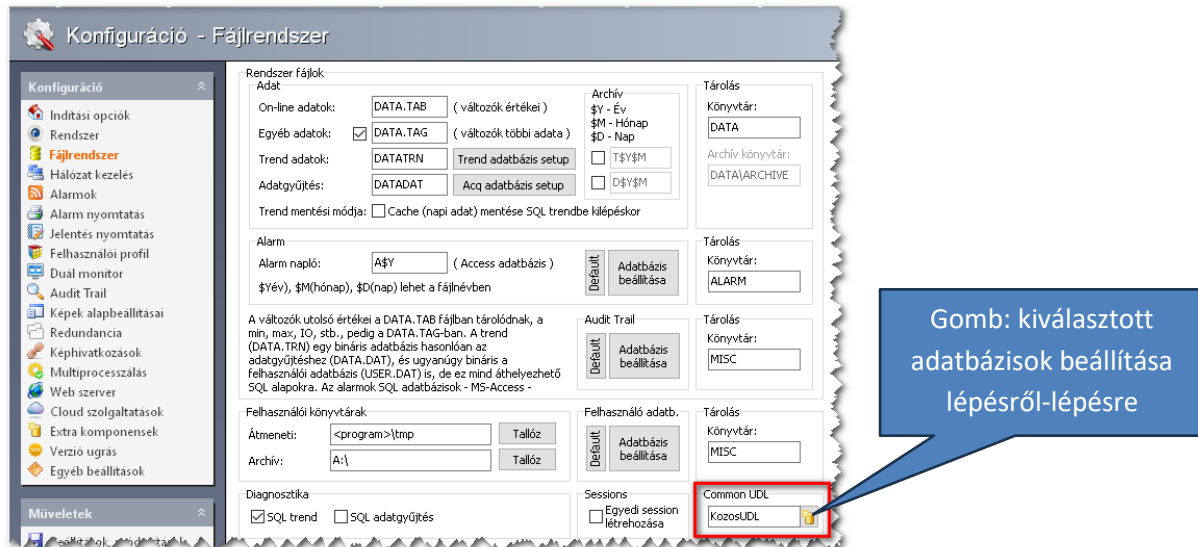
Függvény értéke	Jelentése
Net server	Redundáns hálózat aktív szervere
Standby server	Redundáns hálózat befigyelő szervere
Local server	Elosztott hálózati rendszerek lokális csomópontjai, amelyek szerverként futnak a distribution server felügyelete alatt
Server	Normál szerver csomópont
Client	Normál kliens csomópont
-	Nincs hálózat

A késsel megszínezett sor felel meg az előző fejezetben vázolt esetnek. Ez a csomópont típus szerver VISION szempontból (lokális adatgyűjtő) és kliens hálózati szempontból, ami *distribution service*-en keresztül a Fact-IoT központi szerveret táplálja és teszi elérhetővé annak adatait a Data Lake-en belül más munkaállomások számára is. Régebben a DFS hálózattal konfiguráltuk az ilyen eseteket.

X10 Verzió 4.5

1. Közös adatbázis kezelő UDL(x) fájl

A 4.5 verziótól kezdve nem feltétlenül szükséges minden egyes SQL adatbázishoz külön leíró készíteni, ha ugyanabban az adatbázisban szeretnénk tárolni a trendet, az adatgyűjtést, az alarmokat az Audit Trail adatbázisát és a felhasználókat is. Csupán egyetlen UDL vagy UDLX kiterjesztésű leíróra van mostantól szükség. Az új konfigurációs opció neve: *CommonUDL*:



A fájlrendszer beállításánál található az új editbox a jobb alsó sarokban – ahogy a képen is látszik. Maga a közös UDL fájl az alkalmazás fájlrendszerében helyezkedik el, a fenti esetben *KozosUDL.udl* néven lesz nyilvántartva. De természetesen tehetjük bárhová – relatívan az alkalmazás könyvtárához képest. Ezzel az új opcióval jelentősen leegyszerűsíthetjük a konfigurációs munkát, hisz alkalmazásonként egy fájlt kell csak módosítani öt helyett!

Van azonban még egy okosság a dologban. Felvetődik a kérdés, hogy mi van akkor, ha csak a trendet és az adatgyűjtést szeretnénk SQL-ben tárolni, a többi adatbázist továbbra is a defaultban. Például azért, mert azok más SQL adatbázisban vannak vagy nem is kívánjuk őket SQL-be áttenni. Erre a dilemmára a következő egyszerű, bár elsőre talán meglepő megoldás született:

Abban az esetben, ha a közös UDL fájl neve aláhúzással kezdődik, akkor az elnevezése már nem tetszőleges. Úgy kell őt nevezni, hogy: *_TUDAT*. Azért választottuk ezt a nevet, hogy könnyen meg lehessen jegyezni. Mármint minden egyes betűnek szerepe van:

- T: Trend
- U: User adatbázis
- D: Adatgyűjtés
- A: Alarm
- T: Audit Trail

Ha bármelyik betű ezek közül hiányzik, pl. aláhúzással helyettesítjük, vagy bármi mással, akkor azt az adatbázist a rendszer nem kezeli közösen. Például a *_T_DAT* elnevezéssel azt érjük el, hogy a

felhasználók kezelése különválnak vagy át sem kerül az SQL adatbázisba. Ha csak a trendet akarjuk SQL-ben tárolni, akkor a közös UDL név: `_T_____` (de lehet akár `_Txxxx` is).

A megoldás további előnye, hogy az alkalmazáson belül tetszőleges számú különböző SQL hivatkozást készíthetünk az eltérő nevű VPR fájlokon keresztül.

Képzeld el, hogy egy alrendszer adatait szeretnénk lokálisan kezelni és egy központi vezénylőben is megjeleníteni. Eddig ehhez két, egymástól eltérő fájlrendszerre volt szükség, hiszen az alrendszer lokális adatbázisai eltérnek a központi adatbázisoktól. Más volt pl. a `TREND.UDL` fájl tartalma lokálisan és más a központban. (Csak cloud adatbázis kezeléssel lehetett a problémát eddig kikerülni.)

A közös UDL fájl projekt fájlunként eltérő megadásával ez a probléma is megoldódik: az alrendszeri és a központi forráskód teljesen azonos lehet. Anélkül másolható tehát, hogy valamiféle különbségekre kellene odafigyelnünk, és mindössze annyit kell tenni, hogy más VPR-t indítsunk a központban, mint az alrendszerben. Mivel a *CommonUDL* megadást a VPR fájl tartalmazza.

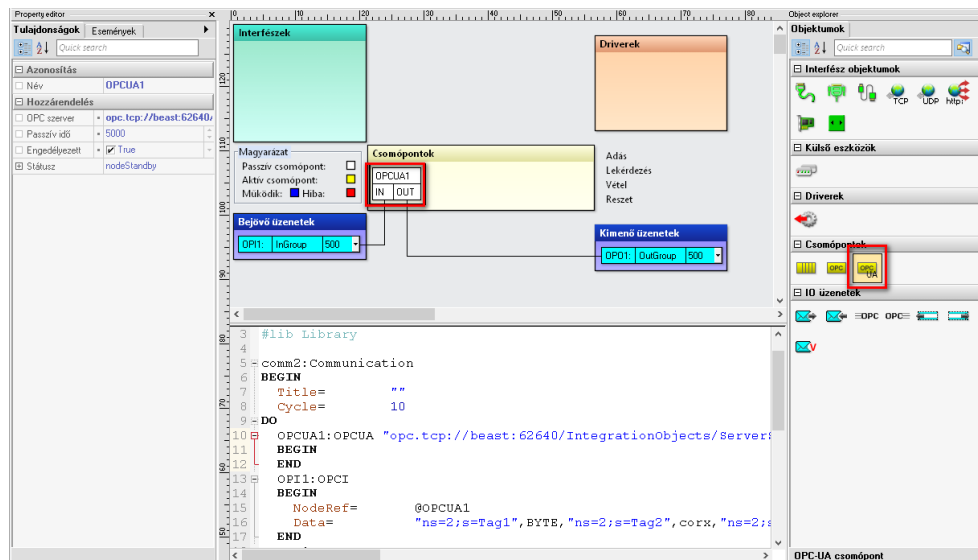
```
Config
StartupImage= "Denso Főkép (Denso_main)"
AutoExec= Yes
MultiWindow= No
AuditEnable= Yes
InterfaceArray=Yes
IndividualTypes=Yes
SQLMessageEnable=No
SQLKeepAlive= Yes
TrendStorageLimit=12
AcqStorageLimit=12
TrendFile= "DATATRN"
AcqFile= "DATADAT"
AlarmFile= "ASV"
CommonUDL= " T_DAT"
EnableTag= Yes
TrendSQLStatistics=Yes
IsMainConfig=3
```

X10 Verzió 4.4

1. OPC-UA kommunikáció

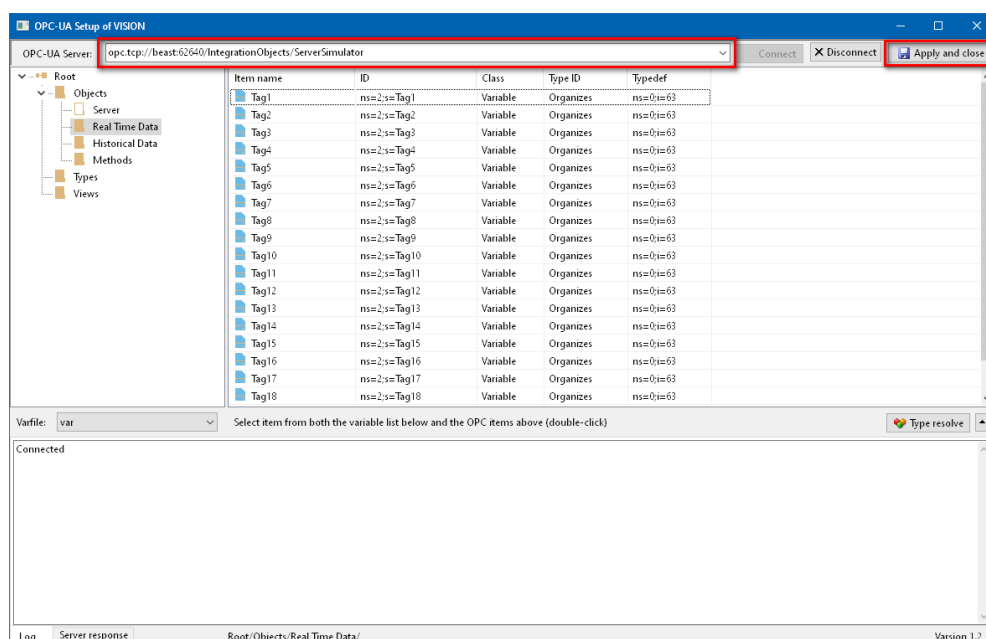
A 4.4 verziótól kezdve elérhetővé vált az OPC-UA (*Universal Architecture*) kommunikációs rendszer, ami az OPC-DA kiváltására készült. Az OPC-UA-t minden gyártó támogatja. A VISION-be épített változat kivételesen gyors, köszönhetően a legjobb és legkiforrottabb megoldás implementálásának, ami a piacon egyáltalán beszerezhető.

Az OPC-UA is csomópontként jelenik meg a VISION-ben, mint az OPC-DA. Az alábbi képen látható az új objektum típus és egy tesztkonfiguráció:



Nincs különbség a tag-lista hozzárendelését illetően sem – ugyanazok az OPC input és output struktúrák (objektumok) használhatók itt is, csak értelemszerűen más lesz maga a lista.

Amikor az OPC szerverre kattintunk (ami önmagában jóval egyszerűbb, mint az OPC-DA szervere, hiszen nincs pl. XML gateway), akkor a következő ablak jelenik meg:



Mindössze annyit kell tenni, hogy kiválasztjuk az OPC-UA szervert, majd *Apply and close*. Ha már létezik, vagy valaha beírtuk a megfelelő kapcsolati stringet, a legördülő menüben szerepel; ha nem, akkor be kell gépelnünk. A képen látható `opc.tcp://beast:62640/Integration....` string egy szimulációs szervernek az elérhetősége a *beast* nevű gépen, 62640-es porton. De természetesen itt a valódi IP címet és portot kell megadni:

`opc.tcp://ip cím:port cím ( például: opc.tcp://192.168.3.12:67512 )`

Általában ennyi kell csupán, néha komplikáltabb kissé az OPC-UA server URL-je, de ezt az adott környezetben megadják. Kipróbálni a *Connect* paranccsal lehet. Ha a kapcsolatfelvétel sikeres, megjelenik az első szintig kifejtett OPC-UA fastruktúra, ahogy a képernyő fotón is látjuk. Innen a szokásos módon mehetünk tovább. Az ikonok az OPC-UA elem típusát jelzik – például a sárga a mappának felel meg, ami tovább nyitható dupla kattal –, de az OPC-UA esetén nincs korlátozva, hogy az alaphelyzetben végpontnak számító teli kék ikon ne lenne tovább nyitható. A tényleges taglistát meg kell keresnünk.

Eltérően az OPC-DA-tól, az OPC-UA struktúrák jóval komplikáltabbak és sokkal több típus található meg bennük, nem csak tag-ek. Ezért nem bontjuk ki alapból a teljes struktúrát. De ha kíváncsiak vagyunk rá, kibontogatjuk, amit lehet, és könnyen a jobb oldalon látható struktúráig jutunk még a tesztprogramban is. A végpontok száma könnyen lehet több ezer!

Ugyanakkor a tag-ek általában egy *Objects* csoport alatt találhatók, elég csak azzal foglalkoznunk. Hogy tudjuk-e olvasni az adatokat, már ezen a konfigurációs képen kiderül: egyszerűen kattintsunk duplán a végponti tag-re. Ha olvasható, akkor a kép alján látható logban elolvashatjuk az értékeket:

Item name	ID	Class	Type ID	Typedef
Tag1	ns=2;s=Tag1	Variable	Organizes	ns=0;i=63
Tag2	ns=2;s=Tag2	Variable	Organizes	ns=0;i=63
Tag3	ns=2;s=Tag3	Variable	Organizes	ns=0;i=63
Tag4	ns=2;s=Tag4	Variable	Organizes	ns=0;i=63
Tag5	ns=2;s=Tag5	Variable	Organizes	ns=0;i=63
Tag6	ns=2;s=Tag6	Variable	Organizes	ns=0;i=63
Tag7	ns=2;s=Tag7	Variable	Organizes	ns=0;i=63
Tag8	ns=2;s=Tag8	Variable	Organizes	ns=0;i=63

Következik az adat lista, vagyis a VISION változók és az OPC-UA itemek összerendelése. Ez ugyanazt a mintát követi, mint az OPC-DA esetén – apró eltérésekkel:

Variable name	Description	Value
BYTE	Sample byte variable	
INT	Sample integer variable	
cory	Sample integer X-coord	
cory	Sample integer Y-coord	
angle	Sample integer angle	
rea	Sample real variable	
imenu	Sample integer menu-var	
menu	Sample integer menu-var	
sendflag	Sample enable sending	
bit	Sample bit-variable	

Az itt látható ablakot az OPC-I – bemenet, ill. OPCO – kimenet objektumokra való dupla rákattintással kapjuk; vagy az *Adat* tulajdonság ikonjára kattintva, ahogy opc-DA esetén.

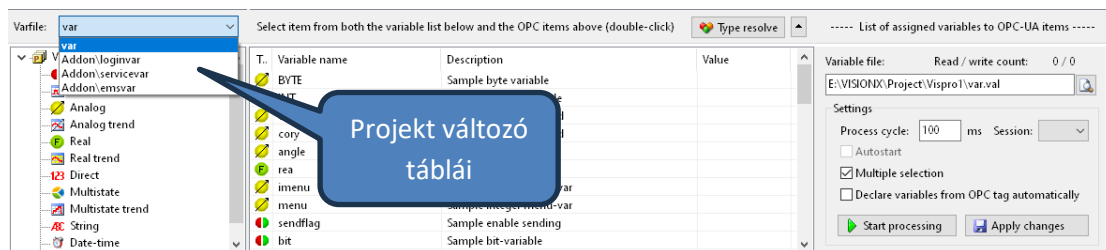
Az összerendelés is hasonló: az ablak jobb felső mezőjébe kell dupla kattintással, vagy drag-&-drop művelettel elhelyezni egyrészt az OPC elemet, másrészt a kép alján látható VISION változólistából a változót. Az item elnevezése gyakran ilyen: `ns=2;s=DataTag`, vagy `ns=2;i=1892`. Mindkető azonosítja az adatot, hiszen egyrészt megadja a node session-t (2), másrészt a tag konkrét nevét (s=), vagy indexét (i=).

Maga a program számos okosságot tartalmaz:

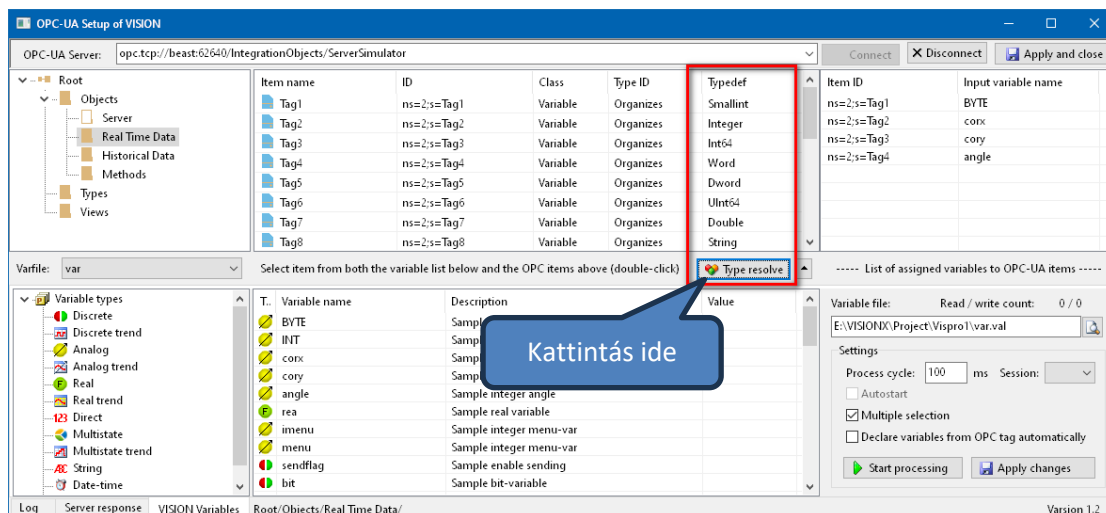
- Az összerendelési listában mozogva a program az OPC itemek és a változók között is megmutatja, hogy az adat mely két elemből tevődik össze
- Kipróbálhatjuk a *Start processing* gomb megnyomásával, hogyan működne az adott összerendelési lista ciklikus olvasása adott ciklusidővel (eredmények a log fülön, alul)
- A log földre visszakattintva olvashatjuk (dupla katt) és **írhatjuk** is a változót; a művelet eredménye látszik a logban (nem biztos, hogy van jogunk írni egy változót!)
- Kontrollal és shifttel több elemet is összeválogathatunk az OPC elemlistában, amit azután egyetlen mozdulattal (drag=&-drop) vihetünk át az összerendelési listába
- Automatikusan felvehetjük a tageket, mint VISION változókat a VISION változólistájába

Ez utóbbi funkciót használva pár egérr kattintással tudunk komplett működtető adatbázist készíteni bármilyen OPC-UA alrendszerhez. Ehhez a *Declare variables from OPC tag automatically* opciót kell checkkolni csupán. Ha ráadásul a VISION Fact-IoT platformját is üzemeltetjük a háttérben, még az alkalmazást sem kell újraindítani. A feladat mindössze a következő: (1) összes kijelölése (Ctrl-A), (2) átmozgatás az összerendelési listába, (3) mentés. Innen kezdve minden változó létezik és máris olvashatjuk / írhatjuk.

Ne felejtsük el, hogy több változótábla esetén ki kell választanunk a megfelelő változó fájlt (táblát):



Végül még egy hasznos szolgáltatásra hívnám fel a figyelmet: változók típusának a leolvasására:



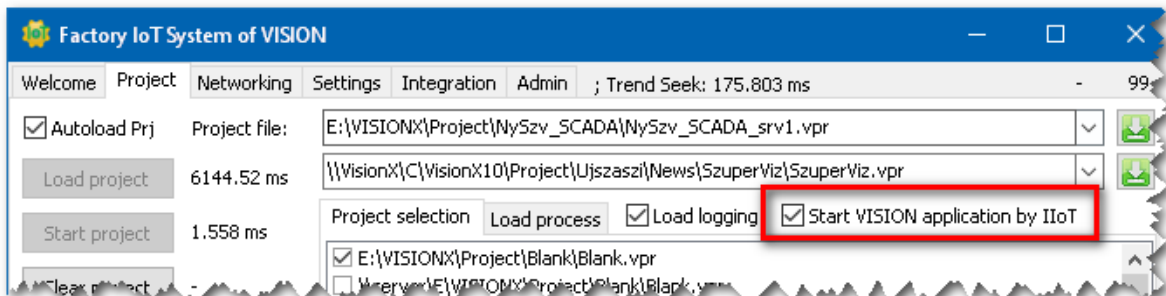
Ezzel kapcsolatban azt kell tudni, hogy az item-ek lekérdezésekor az OPC-UA még nem tudja, hogy milyen típusú az adat. Ahhoz az adatot be is kell olvasni. Ezért ezt nem csináljuk meg automatikusan, hiszen tagból egy listában lehet több ezer, csak külön parancsra. A beolvasás mindig szövegstring formájában történik, így a program minden lehetséges módon *cast*-el. Ám a szövegváltozókat, a boolean típust és a numerikust mindenképp meg kell különböztetni egymástól – gyakran az egészt és a lebegőpontost is. Egyebekben a típus a számbázolás értéktartományát befolyásolja.

2. IIoT módosítások

A VISION-be ágyazott Fact-IIoT platform új változata 1.21-es verziószámmal rendelkezik. Három változtatás történt:

- Elérhetővé vált a *NumVision* függvény az IIoT taszkokon belül is
- Jelentősen felgyorsult az SQL alapú trendeket tartalmazó alkalmazások betöltési ideje (50-szeresére)
- Egy új opcióval lehetővé vált, hogy a VISION alkalmazást maga a Fact-IIoT szerver töltse be, miután ő maga elindította a projektet.

Ez utóbbi módosítással az IIoT-VISION szinkronizációnak egy sokkal egyszerűbb és biztosabb módja vált lehetővé, hisz a VISION már biztosan akkor indul csak el, ha az IIoT platform működik. Duplaszerveres klienseken különösen hasznos az új szolgáltatás, mivel oda elég mostantól magát az IIoT szervert feltelepíteni (egy exe és az ini file), majd azt elindítani. Az IIoT szerver kikeresi az aktív redundáns szervert (NetServer) és ott indítja el az alkalmazást. Az új opciót mutatja az alábbi screenshot:

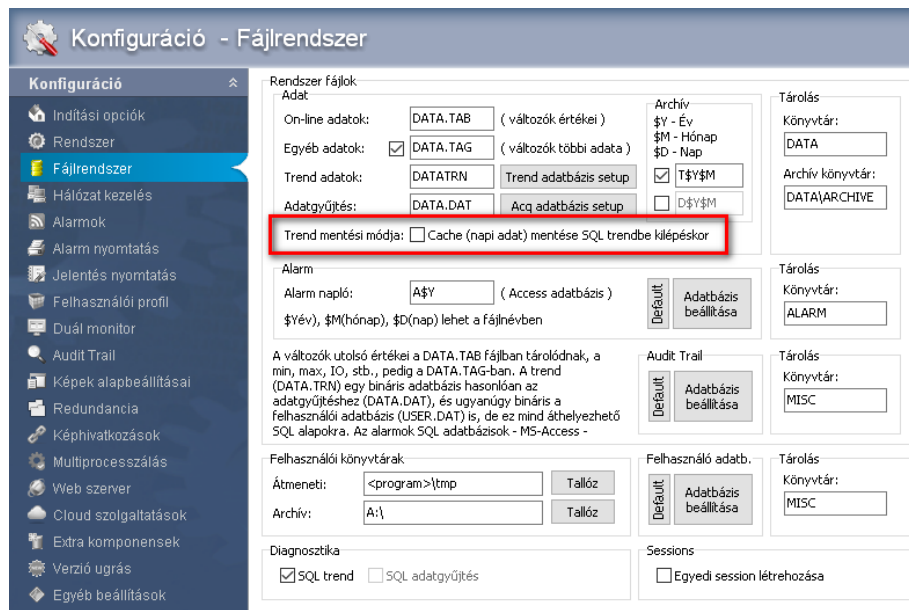


Egy apróbb hibajavítás is történt, bár az nem az IIoT-t, hanem a VISION-t érinti. Napváltás után a VISION alkalmazás képek közötti váltása lelassult egy inicializálatlan *mutex* miatt, de szerencsétlen módon csak IIoT-val együtt. A hibát a VISION-ben (4.4) javítottuk.

3. SQL trend mentés kikapcsolása VISION-ben

Itt lényegében a Fact-IIoT-ben bevált gyakorlatot tettük át a VISION-be is. Emlékeztetőül: nagyszámú SQL trend esetén eltart egy ideig – akár több percig is –, mire a VISION a cache teljes tartalmát átírja az SQL trend-adatbázisba, amikor kilépünk a programból. Ugyanakkor általában feleslegesen, mivel az adatokat a napi bináris adatbázisba (DATATRN.TRN) már kimentettük a kilépés pillanatában, és újraindítás után innen olvassa vissza a VISION a teljes napi rekord tartalmát. Adat tehát nem vész el azzal, hogy az SQL trendet kilépéskor nem frissítettük. Hacsak nem pont éjfélkor állítjuk le és indítjuk újra a VISION-t. Ezt a szituációt kerülni javasoljuk.

A megfelelő konfigurációs oldal látszik a következő ábrán (konfiguráció – fájlrendszer):



Alaphelyzetben a *cache (napi adat) mentése SQL trendbe kilépéskor* checkbox aktív. Kikapcsolása az összes projektre hat, mivel a változtatást a program a vpr.ini fájlba írja.

4. Trend konvertálása XDB (CSV) fájlba

Az XDB export nem új funkció, hisz a *TrnCsvExport* függvény is erre szolgált. Alapvető különbség azonban az új *TrnXdbExportEx* függvénnyel összehasonlítva, hogy az előbbihez kellett maga a trend is, ami az adott trendváltozókat ábrázolta, lévén a *TrnCsvExport* magából a trend objektumból olvasta ki az adatokat. Az utóbbi függvény azonban közvetlenül a trend adatbázisból, ezért akár háttértszken, pl. ciklikus programban is kiadható, hogy pl. naponta kiexportáljunk a trendből bizonyos adatokat.

A prototípus deklaráció:

```
TrnXdbExportEx:: procedure (xdbfile: namestring; header: ushort;
                             timefrom: long; timeto: long;
                             resolution: long; vars: list of univ)
```

Az eljárás végén kell megadni a konkrét trendváltozó listát, a felbontás (*resolution*) másodpercben értendő. A *timefrom* és *timeto*, vagyis az időtartomány pedig packtime formátumban. A *header* integer típus, értékei határozzák meg, hogy a fejlécbe mi kerül át a változóból:

Header=0	változó név
Header=1	változó leírása
Header=2	változó Smart azonosítója (ezt használjuk az energiamenedzsmentben)

Végül a változó lista helyett megadható a *Varlist.xdb* is, hogy az Addon objektumok által várt formátumban adjuk át az előszelektált adatokat a függvénynek is – megkönnyítve ezzel az összeválogatást, konvertálást.

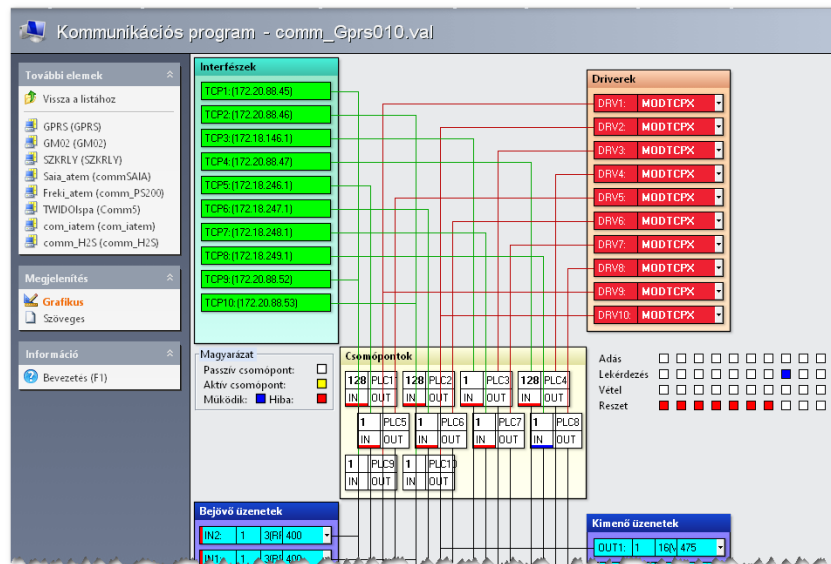
Példa:

```
TrnXdbExportEx ("Trend1.xdb", 2, cfrom, cto, 10, "Analytics\VarList1.xdb");
```

X10 Verzió 4.3

1. Új MODTCP driver

Az új driver neve: MODTCPX. Kifejlesztését a nagyszámú Modbus TCP végpont használata indokolta, amelyet gyakran az alábbi ábrán látható módon szolgálnak ki a felhasználói kommunikációs programokban (majd az ilyen taszkokból készül aztán több tucat, sőt több száz):



Megjegyezve, hogy itt elég lett volna 1 db TCP interfészhez (zöld) kötni az összes csomópontot. A beállítás ettől függetlenül helyes – hisz célszerű minden egyes végponthoz külön drivert (piros), ezzel külön szálon futó taszkot rendelni. Nem ezzel van a baj.

Hanem azzal, hogy nagyszámú ilyen taszk esetén egy Windows-ba épített, az összes DLL számára és összméretére vonatkozó korlát lép léphet életbe – nagyjából 100 driver körül –, ami megakadályozza egy ponton túl új driverek betöltését. 64-biten is! A hibát a fejlécbe írt hibaüzenet jelezte:

...has no valid status_service

A MODTCPX driverből azonban kommunikációs taszk szinten töltődik be egy-egy újabb példány – a fenti 10 driver szál helyett csupán 1. A MODTCPX önmagán belül indít annyi szálat, amennyi szükséges. Ezért az egyes csomópontok továbbra sincsenek egymásra hatással. Sőt, **az új megoldás takarékosabb a memóriahasználatot tekintve is és még ráadásul gyorsabb is**. Érdemes használni akkor is, ha egyébként a driver-szám a változtatást önmagában nem indokolja.

Két alapfeltételnek kell teljesülnie a használatához:

1. Minden driver külön csomópontokhoz kapcsolódik
2. Nem kombinálható a módszer a "Connect by channel" opcióval (Modbus TCP – RTU gateway, pl. MOXA esete)

Egyetlen hátrány van csupán és az is csak látvány: ugyanabba a *driver üzenet*be sűríti az összes kommunikáció összes üzenetét a program, hisz a driverből valójában csak 1 példány fut.

2. Fact-IoT módosítások

Több kisebb módosítás került a Fact-IoT szerverbe, amiért mindenképp érdemes frissíteni is (új admin funkciók, Rest-API szolgáltatások, mérőkiolvasó felhősítés stb.).

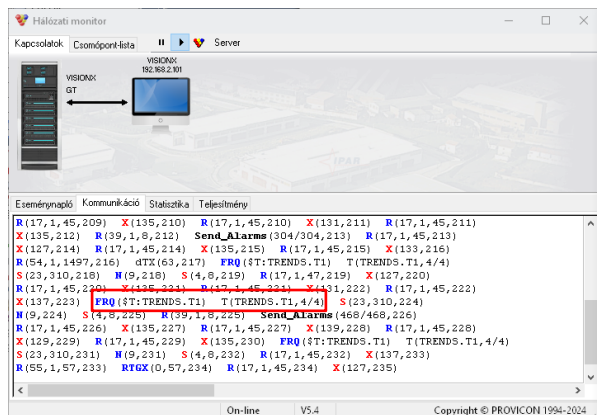
A *NetState* függvényt érdemes külön kiemelni, ami azért érdekes, mert a hálózati funkciók (VISION hálózat, DFS, Web) továbbra is csak VISION szinten léteznek, az IIoT-ban futó szervízek eddig nem látták annak állapotát, hiszen egy másik szerverben futottak. A VISION X10 4.2 és Fact-IoT 1.20 verzió-kombó azonban megoldja ezt a problémát: amennyiben a *NetState* függvény feldolgozása az IIoT szintjén történik, a függvény mostantól kiértékeli a VISION-ból kapott információkat.

X10 Verzió 4.2

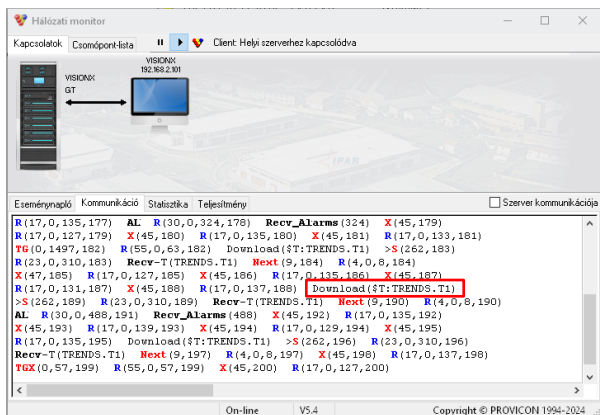
1. Trendek frissítése SQL trendeknél

Normál hálózati környezetben a trendek frissítése úgy történik, hogy a kliens az általa épp megjelenített trend(ek) frissítését kéri a szervertől – meghatározott időközönként. A szerver ennek hatására a memóriában tárolt (cache-elt) mintákat az SQL adatbázisba kimentti, a kliens pedig onnan kiolvassa. Egyébként, ha a trendfrissítést a kliens nem kérné, csak 15-60 perc késleltetéssel jelennének meg a friss adatok, lévén alpból ilyen gyakorisággal menti a szerver az átmenetileg memóriában tárolt adatokat az SQL adatbázisba. Ez látható az alanti hálózati monitor képeken:

Szerver oldal:



Kliens oldal:



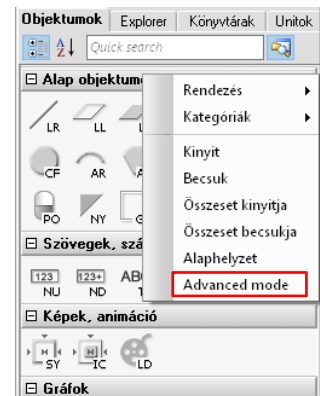
A frissítési időt a program az átlag mintavételezési idő szerint állítja be. Emiatt előfordulhatott, hogy az idő túl hosszúvá nyúlt. **A 4.2-es változatnál ezért a frissítése idő maximumát 1 percre korlátoztuk.**

Megjegyzendő, hogy IIoT platform használatával a probléma nem létezik, ott a trend mindig realtime.

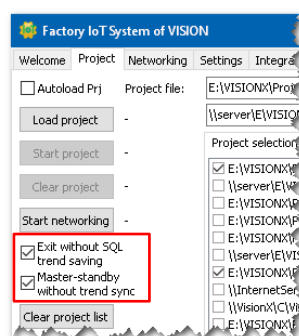
2. Advance mód bevezetése

Az objekt explorer alpból sokkal kevesebb objektumot mutat, mint korábban. Ennek az az oka, hogy bevezettük a *basic* és az *advanced* üzemmódokat az új felhasználók kedvéért, akik elvesztek a sok objektum között. Azonban könnyen visszakapcsolhatod az object explorerre jobb egérgombbal kattintva (ld. jobb oldalt).

Advanced módban az összes többi objektum is láthatóvá válik.



3. IIoT változtatások



Több kisebb rancfelvarrás is történt az 1.10-es változatban, ezért mindenképp érdemes frissíteni:

- A kliens oldali SQL trendek kezelésében történt változás, hibajavítás
- Redundáns üzemmódban kikapcsolhatóvá tettük a DATATR.NRN fájlok szinkronizációját, mivel igazából felesleges, hiszen ebben az üzemmódban lokálisan kezeli a fájlrendszert a VISION, a jelzett adatfájl ezért mindkét oldalon friss. Nézd meg a [videót](#) is.

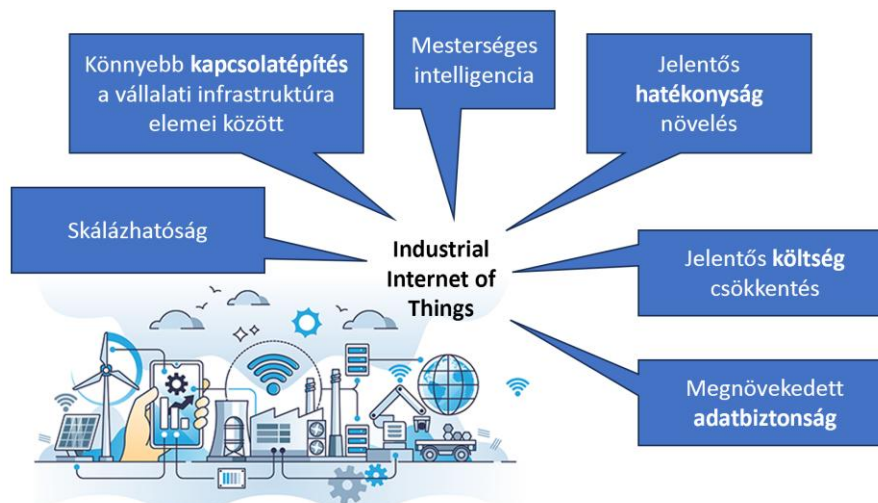
X10 Verzió 4.1

1. IIoT integráció

Az IIoT bevezetése újabb hatalmas mérföldkő a VISION, de igazából az összes *modern* SCADA életében. Mint látni fogjuk, sokkal egyszerűbbé teszi a legtöbb SCADA szintű feladatot (hálózatok, redundáns rendszerek, real-time trendek, módosítás érzékeny átvitel hálózati környezetben stb.), ezért a bevezetése még abban az esetben is indokolt, ha egyébként nem rendszerintegrációra (SCADA + energiamenedzsment + karbantartás menedzsment + gyártás automatizálás stb.) használjuk, amire az IIoT-t eredetileg kitalálták. Hanem csak azért, mert gyorsabban és kényelmesebben szeretnénk dolgozni, meg némi piaci előnyhöz jutni azáltal, hogy a világ élvonalába tartozó ipar 4.0 megoldásokat kínáljuk!

Az IIoT egy exponenciális ütemben bővülő terület, az IoT (Internet of Things) berobbanásának a következménye, hatása az ipari környezetek fejlődésére. Így született meg az IIoT (Industrial Internet of Things), ami egy ipari rendszerekre fókuszáló részhalmaza az IoT világának és az általa indukált technológiáknak. Hisz jellemzően lokális ipari eszközök összekapcsolására és az adatfeldolgozási lánc kiszolgálására kerestek megoldást az ipar által megkövetelt adatbiztonság, elérhetőség és kibervédelem szem előtt tartásával. Az internet e rendszerek esetén csak a legmagasabb szinten kap szerepet, ahol az adatok komplex analízise történik, felhasználva a gépi tanulás és a mesterséges intelligencia módszereit.

Túl a SCADA-ra gyakorolt pozitív hatásain (amiről később részletesebben fogok írni), az IIoT rendszerek a következő fontos területeken eredményeztek áttörést:



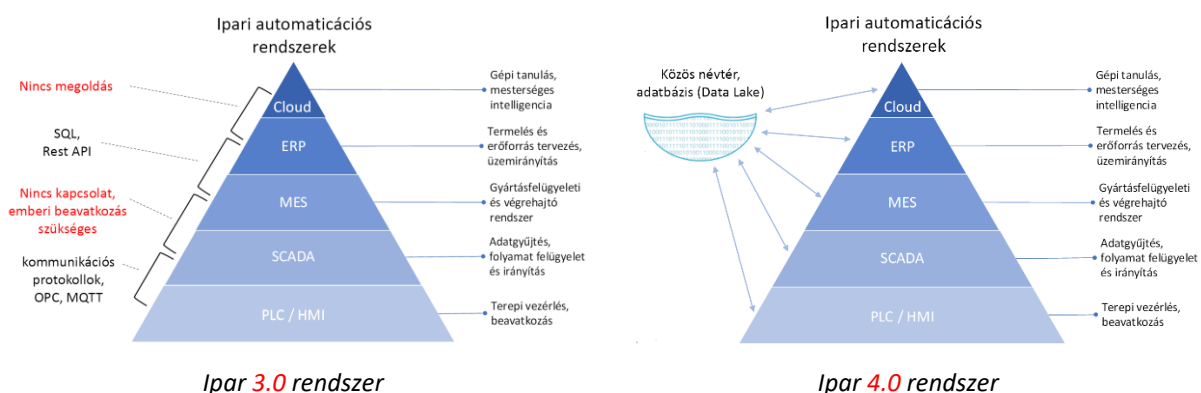
Ezek nem csak szavak. Valóban alkalmazzuk a mesterséges intelligencia vívmányait, sőt kifejezetten emiatt alapítottunk egy külön céget is Ener-Aid néven. Kérem az olvasót, hogy tekintse meg a témával kapcsolatos prezentációnkat és segédanyagainkat! Itt, a fejlesztési dokumentációban csak egy rövid összefoglalóra van lehetőség. A prezentáció:

https://www.provicon.net/presentation/eneraid/Saia_IIoT_presentation.mp4

1.1. Az IIoT megszületése

Mindenekelőtt szögezzük le: **az IIoT az ipar 4.0 rendszerek, így a digitalizáció nélkülözhetetlen kelléke.** A miértje a következő gondolatsorral indokolható:

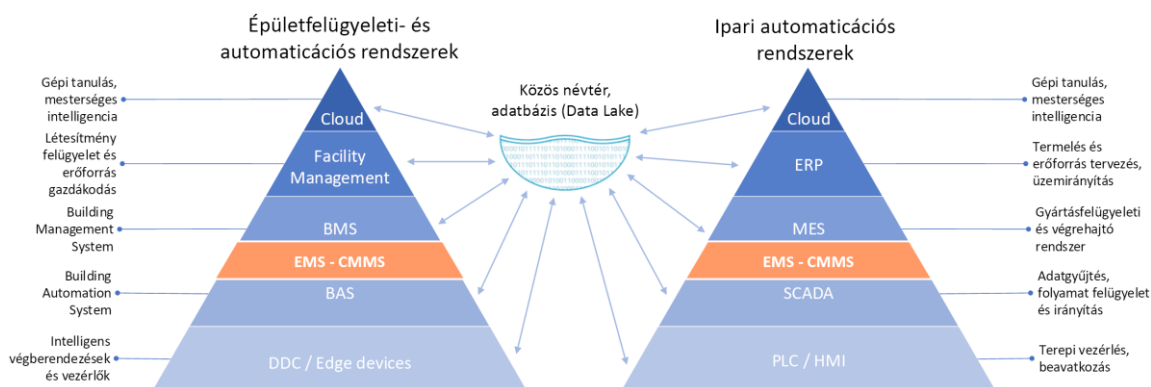
Az ipari *automatizációs stack* elemeit az ipar 3.0 rendszerekben jellemzően egy linerális kommunikációs model kapcsolta össze: a SCADA kommunikál a vezérlő rendszerekkel, PLC-vel, fölötté a MES (gyártás tervezés és végrehajtás) az ERP-vel, a vállalati erőforrás gazdálkodás eszközeivel, ahogy a bal oldali ábra mutatja. Jellemző, hogy nem minden szinttel létezik kapcsolat az ipar 3.0-ban:



Az ipar 3.0 megoldás hátrányai nyilvánvalóak, főleg az alsó szint problémáit ismeri mindenki a dokumentáció olvasói közül: kommunikáció a PLC-vel és az irányítástechnikai / mérés-technikai rendszer elemeivel. Ezt a hihetetlenül munkaigényes és költséges részt egyszerűsítette az IIoT platformra való áttérés, amit az ábra jobb oldalán láthatunk.

Az ipar 4.0 megoldás esetén az összes szint ugyanazzal az egységes névtérrel (*unified namespace*) tart kapcsolatot, s így pl. az sem probléma többé, hogy a MES vagy az ERP szint hozzáférjen a SCADA, sőt a PLC adataihoz. (A MES esetében az OEE számítás miatt ez kifejezetten kívánatos!)

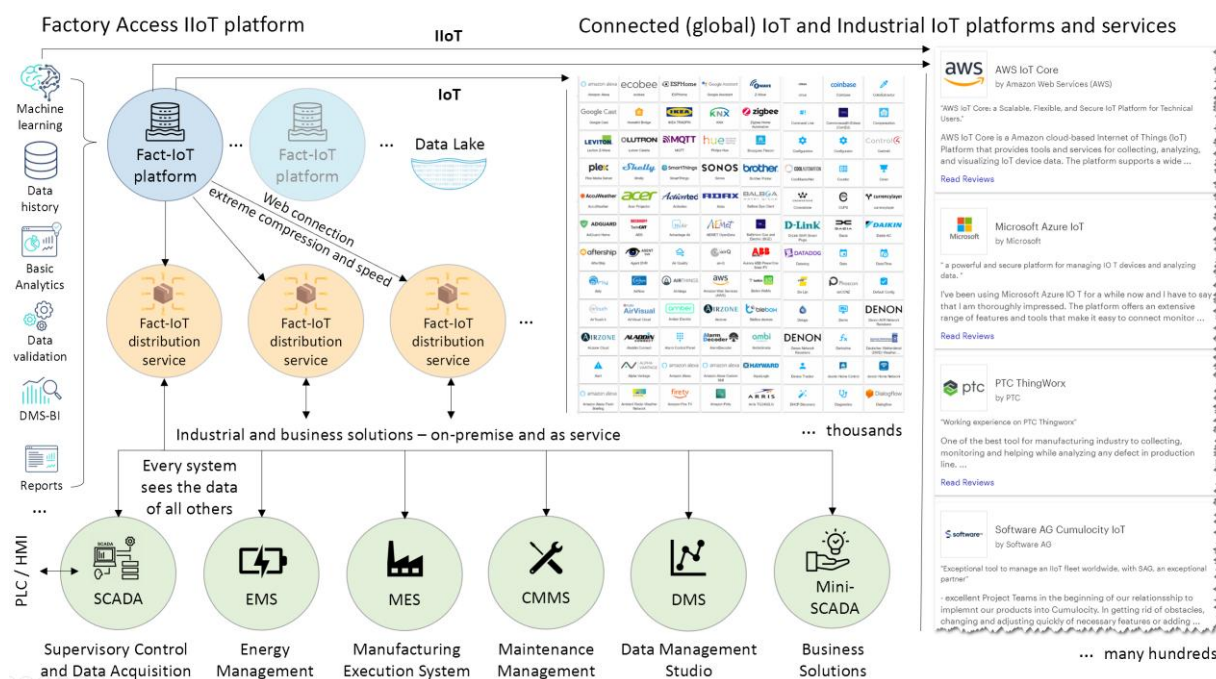
A teljesség kedvéért megjegyzem, hogy az épületautomatikai rendszerek is hasonló problémákkal küzdenek, továbbá az EMS (energiamenedzsment) és CMMS (karbantartás) integrálására is csak az ipari 4.0 kínálja az egyedül értelmes megoldást, mivel az ipar 4.0-ban az EMS is ugyanazon infrastruktúrán keresztül illeszthető – nem kell neki külön mérőhálózatot kiépíteni!



Kell tehát egy adatintegrációs platform (névtér, adatbázis, Data Lake) az ipar 4.0 rendszerhez, amit a jelenlegi fejlesztési trendek az IIoT platformmal azonosítanak. Elvileg lehetne ez SQL adatbázis vagy OPC szerver is, de az meg **jelentősen korlátozná a hozzákapcsolható eszközök körét és az adatok mennyiségét!** Gondoljunk egy villamosenergia mérőre. Egy OPC-UA illesztéshez költséges IIoT edge device-t vagy gateway-t kellene üzembe állítanunk – az egész rendszer végül megfizethetetlen lenne. Olcsóbb, egyszerűbb és könnyebben implementálható megoldásokat keresve a legtöbben az MQTT-t ajánlják. Mi azonban ennél sokkal messzebbre merészkedtünk.

1.2. Fact-IoT

A Fact-IoT egy VISION komponensekre épülő IIoT platform, amit a bevezetőben vázolt alapproblémák megoldására fejlesztettünk. A nevét a gyári hozzáférésre utalva kapta. Vagyis, hogy lokális. Lefelé, a gyári automatizációs rendszerek irányában egy v. több helyi adatbázishoz kapcsolódik. Emiatt nincs a felhasználó kitéve az internet kockázatainak. Felfelé ugyanakkor a Fact-IoT is képes kapcsolatot kialakítani az úgynevezett gyári IIoT rendszerekkel (AWS, Azure stb.), interneten, sőt további IoT platformokkal, mint a Home Assistant. A rendszer lefelé és felfelé egyaránt nyitott:



Érdekessége a megoldásnak a *distribution service* réteg (sárga), ami az alrendszer és a Fact-IoT szerver közé ékelődik és célja az adatforgalom koordinálása és optimalizálása.

Ezzel kapcsolatban azt kell tudni, hogy az IIoT rendszerekben a kapcsolati irány a lekérdezéses rendszerekhez képest fordított, valamint változás orientált. Nagymennyiségű adatot csak akkor lehet egy rendszerrel hatékonyan kezelni, ha a forrás csak a változásokat továbbítja és anélkül, hogy azt valaki lekérdezné, vagyis az egész rendszert körbekérdezné. Azonban ezek a táviratok rövidek, nem igazán tömöríthetőek. A *distribution service*-t közbeiktatva azonban a változások aggregálhatók; összevárva több tíz- száz- vagy ezer üzenetet, már hatékonyan tömöríthetőek. E feladat a Fact-IoT rendszerben olyan jól sikerült, hogy **a tömörítés mértéke akár 2000-szeres is lehet!** Ez az első fontos oka tehát a *distribution service*-nek: a tömörítés (a változásokat továbbító kommunikációs láncban).

Van azonban még egy fontos indok: a nagyterjedésű regionális vagy országos hálózatok támogatása. Egy-egy *distribution service* csak a hozzá kapcsolódó alrendszerek (pl. egy város, vízmű, gyár, épület stb.) adataival foglalkozik, az alrendszernek pedig elég e lokális "adat-koncentrátorhoz" kapcsolódni, mentesítve ezáltal a felesleges kommunikációtól a központot és az országos hálózatot. Végeredményben a Fact-IoT szerver és distribution service szerepek bevezetésével **decentralizált, elosztott, hierarchikusan organizált rendszereket készíthetünk, elképesztően hatékony, tömör és biztonságos adatforgalom mellett.** Ahogy azt a videóban is láthatod, ha megtisztelsz azzal, hogy végignézed.

1.3. VISION implementáció

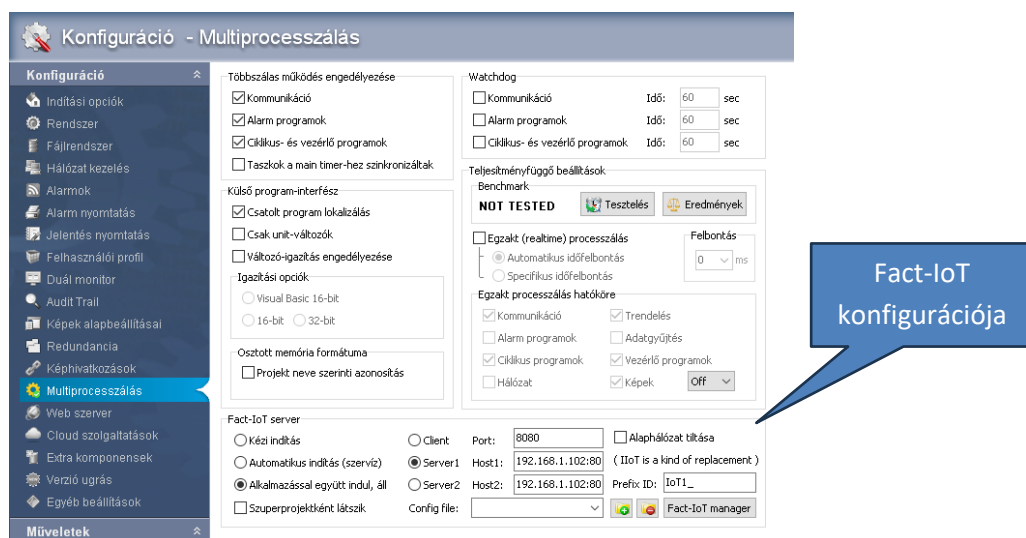
SCADA szempontból a Fact-IoT-re legegyszerűbb úgy tekinteni, mint az új VISION serverre. Az ipar 4.0 ajánlását követve az összes kommunikáció átkerült az Fact-IoT-be. A SCADA nem a PLC-vel tart többé kapcsolatot, hanem ezzel az új szerverrel, ami pedig a PLC-vel.

De itt nem álltunk meg. Az intelligens IIoT platformok képesek scripteket is futtatni, adatot tárolni, alarmot generálni, analitikai feladatokat ellátni stb. Kézenfekvő volt, hogy ezt a VISION esetében se csináljuk másképp: **az összes taszk átkerült a Fact-IoT szerverbe, a VISION-ben csupán a képek és a szimbólumok maradtak – a megjelenítés.**

De továbbra is a VISION-ben szerkeszthető minden, ugyanúgy! Emiatt nem kell félni a váltástól, hiszen minden pont úgy működik, mint korábban. A VISION a háttérben áttölti a változókat, a ciklikus programot, a vezérlőket, az alarmot és a kommunikációt az IIoT szerverbe, de maga szerkeszti azokat.

Persze a Fact-IoT nem merül ki ennyiben, hiszen számos integrációs szolgáltatást biztosít (EMS, CMMS, Node.js, IoT szolgáltatások stb.), ám ezeket külön dokumentációban taglalom. Itt a legegyszerűbb automatizmus ismertetésére szorítkozom csak.

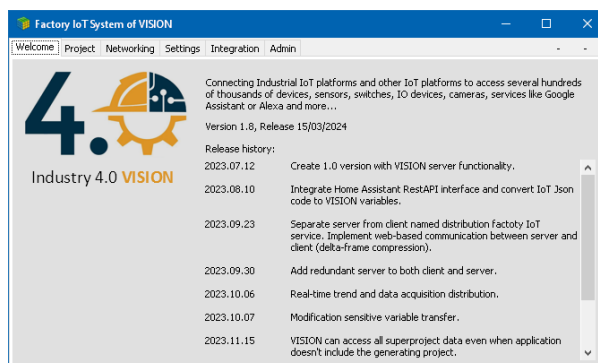
A Fact-IoT automatikus indítását a konfiguráció alatt, a multitaszking menücsoporthoz végezhetjük el



Az új menücsoporthoz a beállító képen alján látható. A legegyszerűbb, ha a képen is mutatott automatikus indítást választjuk (vagy automatikus indítást szervíz üzemmódban). Ekkor az alkalmazás a Fact-IoT-t is elindítja a gépen, lokálisan, processzként. Ez a program lesz a **Fact-IoT szerver**, ha a beállító panelen *Server1*-et vagy *Server2*-t választunk; és **distribution service**, ha a *Client* üzemmódot jelöljük ki. A bevezetőben ismertetett két komponens között valójában a szerver – kliens szerep kiosztásával teszünk különbséget. Ennyire egyszerű.

A Fact-IoT szerver és annak kliense, a distribution service Rest API szolgáltatást használva tart kapcsolatot egymással. Az IP cím és a port beállításával ezt konfiguráljuk be. Két szerveret redundáns környezetben konfigurálunk – ilyenkor két Fact-IoT szerver fut elvileg két, azonos, egymást helyettesíteni képes szerver gépen. A VISION pedig vagy a lokális Fact-IoT szerverrel közvetlenül, vagy a lokális (mindig lokális!) distribution service-en keresztül tart kapcsolatot. A többi opció funkciója a következő:

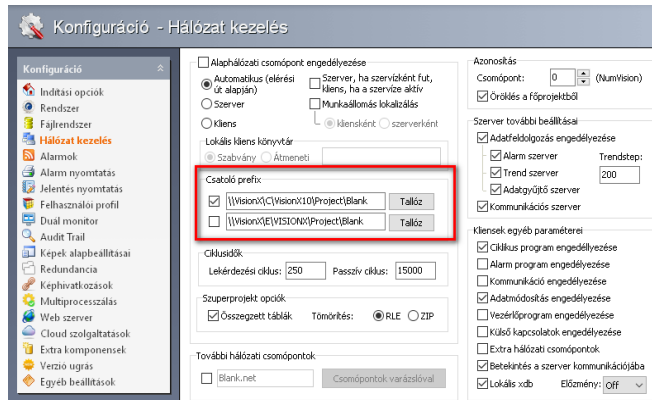
- **Szuperprojektként látszik**, ha az IIoT-be több alkalmazás adatait is feltöltöttük, és ugyanazon vagy egy másik gépen a rendszert megjelenítjük. Ezzel kapcsolatban azt kell tudni, hogy egy projekt képes az IIoT névtéren keresztül más alkalmazásokba is betekinteni, annak változóira hivatkozni, azokat módosítani stb. Bármelyik alkalmazás láthatja bármelyik másik VISION alkalmazás adatait, **közvetlenül**. Ez új! De nem csak az adatokat jeleníthetjük meg a helyi VISION alkalmazásban, hanem akár az összes többi komponenst, képeket, programokat stb. is, mint szuperprojektet. Ezt az üzemmódot engedélyezi v. tiltja az opció.
- **Alaphálózat tiltását**, akkor célszerű kérni, ha az IIoT-t VISION-VISION kommunikációra is felhasználjuk. Az egyszerűsített VISION hálózatra az IIoT használatával nincs szükség többé, hiszen az egységes névtéren keresztül az alkalmazások egymással is automatikusan összekapcsolódnak. Ilyenkor zavaró lehet, ha mellettük a VISION alaphálózat is engedélyezett, hiszen akkor két hálózat üzemel párhuzamosan, feleslegesen, a hazard lehetőségét hordozva. Az IIoT messze gyorsabb bármilyen hálózathoz, akár pár miliszekundumra is lecsökkenthető a ciklusidő, gépen belül pedig már eleve nulla (azonnali). Ergo a VISION alaphálózatot tiltani érdemes: ha az IIoT működik, és engedélyezni, ha nem működik. Ezt csinálja az opció.
- **Prefix ID-t** használva egyedi azonosítóval láthatjuk el a Fact-IoT komponenseit. Ezzel érhetjük el, hogy más még véletlenül se tudjon a rendszerünkhöz hozzáférni (biztonság!), de több lokális példányt is így helyezhetünk üzembe – eltérő azonosítóval –, ha ugyanazon a gépen (pl. tesztelési) célból szervert és klienst, vagy akár több szervert is el akarunk indítani. Így készültek a bemutató videók!
- **Config fájl** rendelhető hozzá a VPR VISION konfigurációhoz. Az automatikusan indított Fact-IoT így aszerint lehet pl. szerver vagy kliens, hogy a VPR-ben melyik ini fájlra hivatkozunk. Amennyiben a szerveren egy SERVER nevű ini fájlt állítunk be, akkor e vpr-t indítva a gép Fact-IoT szerver lesz, ha egy CLIENT nevű ini fájlt, akkor pedig distribution service a klientsen. Pont ugyanazokat az elveket használva csinálhatunk VISION-VISION hálózatot, mint korábban, csak immár ipar 4.0 rendszert készítve! Új konfigurációt a zöld plusz gombbal adhatunk hozzá a rendszerhez, a piros mínusszal pedig törölhetünk. Hogy az adott VPR-be mit konfigurálunk be, a legördülő menü választásától függ.
- **Fact-IoT manager** gombot használva megismerhetjük és megváltoztathatjuk a szerver többi paraméterét: az integrációs platformot, a védelmi token, az admin felületet, a projektek listáját és még nagyon sok mindent. A Fact-IoT manager emellett monitorozás céljára is felhasználható. De mindez külön dokumentáció tárgya.



(A Fact-IoT menedzselésére valójában külön admin web alkalmazás készül, ezért azt itt megmutatott program csak a gyors beállításokat hivatott biztosítani és a folyamatokat monitorozni. Az admin felület egy hosszabb fejlesztési folyamat része, jelenleg még nincs kész teljesen. Viszont a Fact-IoT már most, nélküle is teljes értékűen használható.)

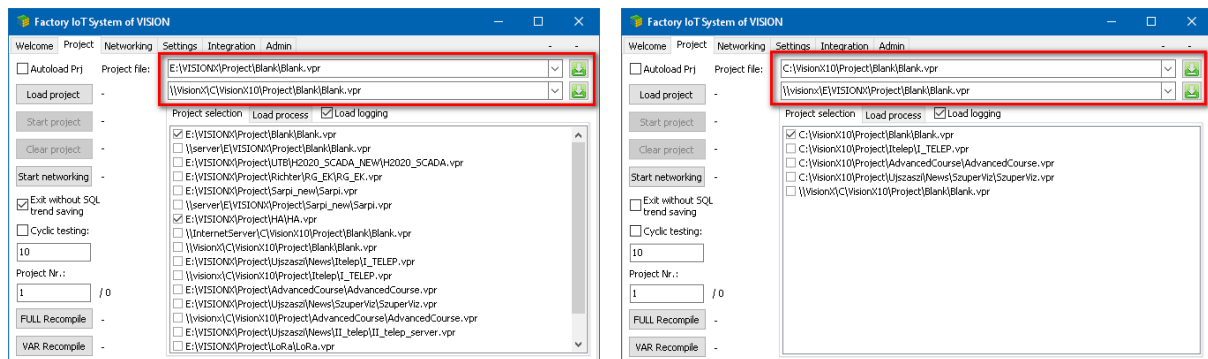
1.4. Fact-IoT redundáns környezetben

A VISION oldalon nem változik semmi, továbbra is be kell állítani a két szerver hálózati elérését a szokásos módon, ahogy alul az ábrán is látszik. Sőt a prioritások beállítása sem változott (checkbox az elsődleges csatoló prefix előtt). A hálózatot pedig vagy kikapcsoljuk, mint a példán (*Alaphálózati csomópont engedélyezése*), vagy az előző pontban ismertetett *Alaphálózat tiltása* opciót használjuk a Fact-IoT beállításoknál. A lényeg, hogy a két hálózat (VISION, IIoT) ne legyen egyszerre engedélyezve.



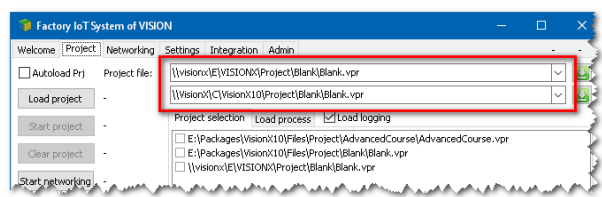
Ezután a Fact-IoT konfigurációnál be kell állítani a *Server1*-et az egyik gépen, *Server2* a másikon. A Fact-IoT esetén ezt konkrétan meg kell határoznunk, a szervereket ui. a Fact-IoT nem keresi induláskor - ellentétben a VISION-nel -, most eleve pontosan tudjuk, hogy melyik melyik. Itt utalnék vissza arra, hogy egy rendszerben több Fact-IoT konfiguráció is létezhet és a VPR-ben hivatkozhatunk rá, hogy melyiket indítsuk.

A másik lényeges különbség, hogy a szerverek mindig a **lokális gép fájlrendszerét használják, a standby server is**. Ez megint különbség a VISION-höz képest, ahol a standby még az aktív szerverre csatlakozott – az egyszerű VISION hálózat szabályait követve – hálózati megosztáson keresztül. A Fact-IoT ezzel szemben mindig két szervert futtat a hozzá tartozó lokális fájlrendszeren, függetlenül attól, hogy NetServer vagy Standby server üzemmódban van. Ezt kell beállítanunk a Fact-IoT manager *Project* fülén:



Ez nem link prefix, hanem annak a konkrét vpr fájlnek az elérési útja, ami a projektet azonosítja. **Az első sor mindig a lokális fájlrendszeren belüli vpr, a második pedig a távoli gépen a neki megfelelő (szerver v. superprojekt) vpr.** A két oldalon ez nem ugyanaz, hiszen egymáshoz képest felcserélt. Macerásnak tűnik, de a szabály egyszerű: **a projekt fájl első sorában mindig azt a programot azonosítjuk, amit a program konkrétan el is indít** (a második sor szerverváltás során kap szerepet).

Végül a redundáns szerverekre kapcsolódó kliens is eltér mindkét szervertől – ott csak hálózati elérések szerepelnek. És persze a beállítás: *Client*. Végeredményben három különböző Fact-IoT konfigurációt kapunk.



1.5. Fact-IoT használatának előnyei

Ezekből jó sok van, ha csak a SCADA-re gyakorolt pozitív hatásokra koncentrálnunk is:

- Tisztán web-es hálózat a Fact-IoT komponensei között http, vagy https üzemmódban, annak minden biztonsági előnyével. A kapcsolat módja: Rest API
- Rendkívüli tömörítés (akár 2000-szeres) és hálózati sebesség (pár miliszekundum)
- A fentiből adódóan minden korábbinál kisebb adatforgalom
- Nagyságrendekkel több adat kezelésének a lehetősége
- Real-time trendek (a kliensnek nem kell többé kérnie a szervert, hogy mentse az SQL adatbázisba az cache-elt adatokat, mivel a cache is látható az egységes névtéren keresztül)
- Változás érzékeny küldés kliensből sokkal gyorsabban és megbízhatóbban, mivel az IIoT-nek eleve a változásérzékeny küldés az alapja
- Nem kell újraindítani a VISION-t változó módosítás miatt, sőt e változások a hálózaton is úgymond "átmennek"
- Az alkalmazások frissítése egyszerűen a fájlrendszer felmásolására redukálható (azonnali, perces, órás vagy napi szinkronizáció mellett)
- Redundáns alkalmazások *net server* <-> *standby server* átváltása miliszekundumok alatt
- Sokkal gyorsabb változó felvétel akár óriás alkalmazáson belül is
- Sokkal gyorsabb lett a unitok hozzáadása, törlése és módosítása is
- A távoli (DFS) rendszerek változótábláját nem kell azonossá tenni a szerverével
- Cloud adatbázis kezelés VISION-nel kompatibilisen, de gyorsabban és védettebben
- A kommunikáció és a taszkok működése is megjeleníthető a VISION alkalmazásban, betekintést adva a Fact-IoT szerver belsejébe (mivel valójában ott futnak)

Ehhez jönnek hozzá mindazon előnyök, amit a Fact-IoT, mint **integrációs platform** nyújt. Például nem kell többé azzal bajlódni, hogy adatokat adjunk át egy VISION alkalmazásból az energiamenedzsment rendszernek, vagy a karbantartás menedzsernek, mivel azok közvetlenül hozzáférnek az IIoT névtéren keresztül az alkalmazáshoz, és megfordítva, a SCADA képernyőn is közvetlenül hivatkozhatunk bármely EMS-ben kiszámított KPI mutatóra, hatékonysági adatra, hozzáférünk az EMS trendekhez, jelentésekhez, dashboardokhoz stb.

A Fact-IoT ezen kívül önmagában tartalmaz mesterséges intelligencia algoritmusokat (szennyvíz technológiára fejlesztünk jelenleg ilyeneket, valamint fogyasztás optimalizálására MES rendszerekben). Tartalmaz ezenkívül gépi tanulást, rendelkezik elemzési képességekkel és tud kapcsolódni további (bolti) IIoT rendszerekhez, mint az AWS vagy az Azure – csak az AWS-en belül további több száz további funkciót biztosítva –, az IoT kapcsolaton keresztül pedig lényegében a világ összes IoT eszközéhez, amiből állítólag ma már több százmillió van! Szóval egy hatalmas világ nyílik ki előttünk...

Van azonban olyan integrációs szolgáltatása is, amit előnyei miatt a VISION-ben közvetlenül is elérhetővé tettünk: Node.js platform. Erről szól a következő fejezet.

2. Node.js integráció, JavaScript programozás

A 4.1 verziótól kezdve már nem csak XSDL-ben, Pascal-ban és C-ben programozhatjuk a VISION-t, de JavaScript-ben is. Nem is akárhogy! A VISION új programozási környezete nem egyszerűen csak szintaktika, ezen a ponton valóban elérhető a JavaScript teljes eszköz- és függvénytára, ráadásul mindjárt a világ legelterjedtebb és leghatékonyabb programkörnyezetébe, a Node.js-be ágyazva.

A Node.js jelenleg a legnépszerűbb programkörnyezet a világon, a fejlesztők 42%-a választja. Elterjedtségben megelőzi a Pythont (Django), a Ruby-t, a Java alapú Spring-et, ... a sorban messze lemaradva kullog az ASP.NET C# valahol sorban a 6-ik hely környékén (ld. ChatGPT). Ez az állapot legalábbis jelenleg, 2024 elején.

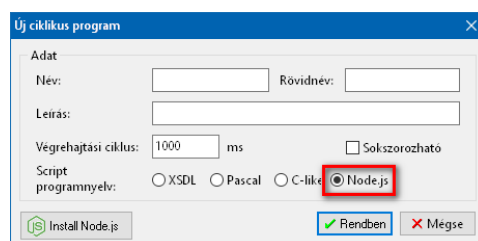
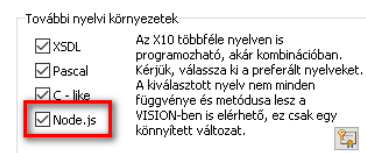
Az okok között a széles támogatottság és a több száz modul áll, ami lényegében minden feladatra megoldást kínál. A rendszert a Google V8 motor hajtja, ami megdöbbentően hatékony. Saját méréseim szerint a C++ gépi programok sebességét reprodukálja, néhány teszten egy 10-es faktort vert a .NET-re.

A Node.js ún. *non-blocking* környezet, ami lényegében az időigényesebb műveletek (fájl, SQL stb.) aszinkron végrehajtását takarja – még a leghosszabb fájlművelet sem képes megakasztani a Node.js végrehajtó programját, és így a VISION-t sem. Mindezt ráadásul úgy, hogy a backend feladatok párhuzamosítása a háttérben futó szálakat sem szaporítja, így a memóriát és az op. rendszert sem terheli túl. Mindez persze főleg nagyszámú, sok párhuzamos ügyfél kiszolgálása esetén előny, amit a VISION-ben jelenleg (*még*) nem tudunk kihasználni. De a jövő kihívásaira fel kell készülni.

Felvetődik a kérdés, kik használják? A felhasználók között olyan neveket találunk, mint a NASA, Airbnb, PayPal, LinkedIn, Netflix, Uber, Walmart stb. **Konkrétan 98%-a a Fortune 500 cégeknek a világon a Node.js-re alapozza a szerver bázisát!** Mostantól a VISION felhasználók is csatlakozhatnak a programozók e népes táborához.

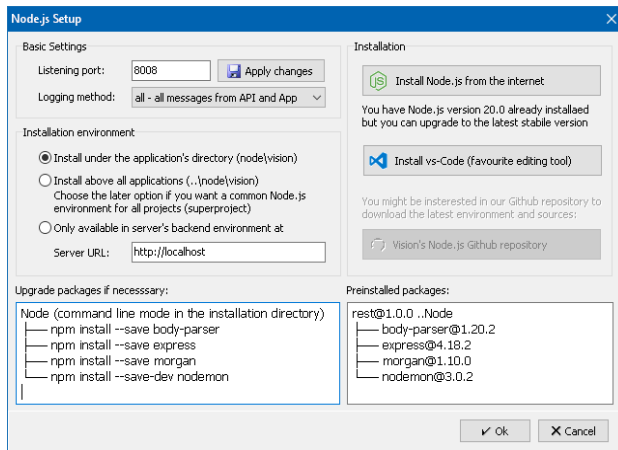
2.1. Node.js engedélyezése és használata

A dolog egyszerűbb, mint az ember elsőre gondolná. Amikor új ciklikus programot készítesz és a többnyelvű programozási környezet is engedélyezett a konfigurációban (ld. jobbra), akkor a rendszer felkínálja a Node.js-t a lehetséges programozási nyelvek között:



A dialógus egyben felkínálja a Node.js motor feltelepítését is. A környezet pár másodperc alatt felmegy bármilyen gépre, és ha nem a VISION szerveren futtatjuk, akkor a rendszer függetleníthető magától a Windows-tól is, mivel a Node.js platform-független. (Meggjegyzem, ez az egyik ok, amiért elindultunk ebbe az irányba.)

Az *Install Node.js* gomb megnyomása egy újabb dialógust dob fel:



Itt egyrészt elindíthatjuk a rendszer letöltését az internetről, másrészt feltelepíthetünk egyéb hasznos komponenseket, mint a *vs-Code* pl., ami egy szintaktikus editor számos kód bázis (JavaScript, TypeScript, html) és adatformátum (pl. JSON) támogatására. De persze a kód VISION-ben is editálható.

A Node maga bárhová feltelepíthető a gépen és ingyenes. Ahogy az összes kiegészítő is mind. Viszont a beágyazott komponens helyét meg kell határoznunk.

Három lehetőség van:

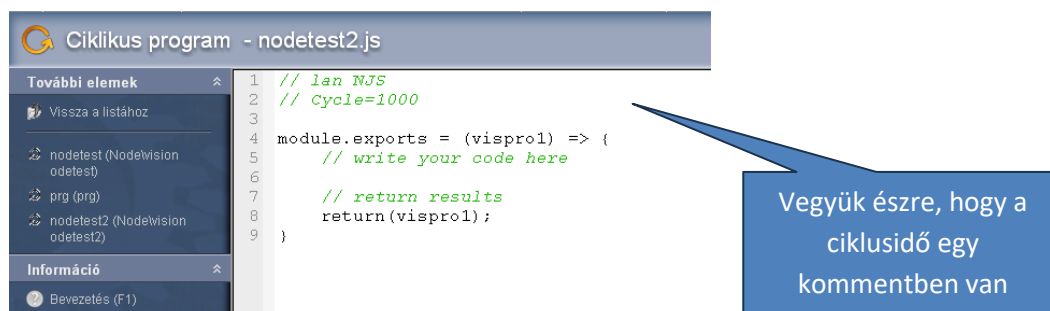
- A Node környezetének telepítése a projekt alá egy *node* alkönyvtárba (*node/vision/prog.js*)
- Egy könyvtárszinttel feljebb, hogy bármely projektből elérhető legyen (szuperprojekt)
- Végül feltelepíthető a VISION szerverétől független gépre is (s akkor nem localhost-on érjük el, hanem a megadott IP címen)

Ez az *Installation environment* box. Meg kell még adnunk ezen kívül a *port* címet és a naplózási módot (*logging method*). Ez utóbbira még visszatérek.

A VISION-be ágyazott környezet egy előre bekonfigurált csomag, ami a legfontosabb komponenseket már eleve tartalmazza (node package manager /npm/, *express.js*, *morgan* és *nodemon*). Ezekkel nem kell bajlódni - a felületen láthatók a verziószámok -, írhatjuk a JavaScript programot.

2.2. JavaScript programozás

Ez már a szokásos eljárást követi, arra azonban ügyeljünk, hogy a VISION által felkínált keretet ne bontsuk meg, ne rontsuk el. Így néz ki a VISION által megírt alap JavaScript ciklikus program:



Magát a programot a két komment közé kell írni (*write your code here...*), a *module.exports* a környezet része és szükséges ahhoz, hogy a programunk megfelelően működjön. A modul egy függvényt hoz létre, ami a *vispro1* nevű struktúra révén a teljes VISION változótáblához hozzáfér. **A *vispro1* a projekt neve, ezért változhat; automatikusan íródik a program forrásába a mentés során.**

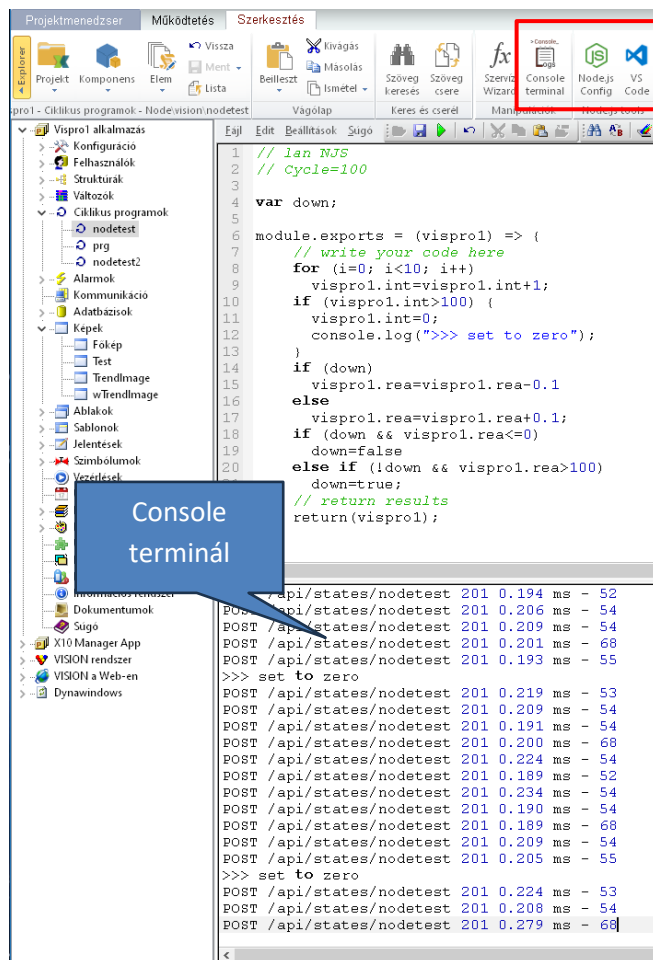
Valójában a változókra is úgy kell hivatkozni, hogy *vispro1.int*, *vispro1.rea* stb. – IIoT környezetben is így hivatkozunk az integrált rendszerek adataira –, de ezeket a kiegészítéseket a VISION megcsinálja, nekünk csak a változónévvel kell törődnünk, ahogy általában is. Egy nagyon egyszerű kis program-részt beírva, majd mentve, a VISION ezt hozza létre:

Minden változó előtt ott van a `vispro1.` referencia, ami szükséges ahhoz, hogy a Node programcskánk a VISION projekt összes változójára hivatkozni tudjon.

A programot debuggolni jelenleg nem lehet – TypeScript-ben van mód rá, ami a JavaScript köré telepíthető –, de ez nem része a jelen leírásnak. **Maga a program akkor fordul újra, ha mentjük!**

Amit viszont remekül használhatunk hibakeresésre, az a `console.log()` javascript függvény, ahogy a példaprogramban is látható. Itt kap szerepet a beállításnál megismert naplózási mód.

```
1 // lan NJS
2 // Cycle=100
3
4 var down;
5
6 module.exports = (vispro1) => {
7   // write your code here
8   for (i=0; i<10; i++)
9     vispro1.int += 1;
10  if (vispro1.int>100) {
11    vispro1.int = 0;
12    console.log(">>> set to zero");
13  }
14  if (down)
15    vispro1.rea -= 0.1
16  else
17    vispro1.rea += 0.1;
18  if (down && vispro1.rea<=0)
19    down=false
20  else if (!down && vispro1.rea>100)
21    down=true;
22  // return results
23  return(vispro1);
24 }
```



Ha jobban megnézzük a szerkesztő ablak menüjét, láthatunk itt három új gombot:

- Console terminal
- Node.js Config
- VS Code

Ezek közül az elsővel jeleníthető meg a console terminál a szerkesztőablak alsó részén. Ugyanezt még egyszer megnyomva, a console eltűnik.

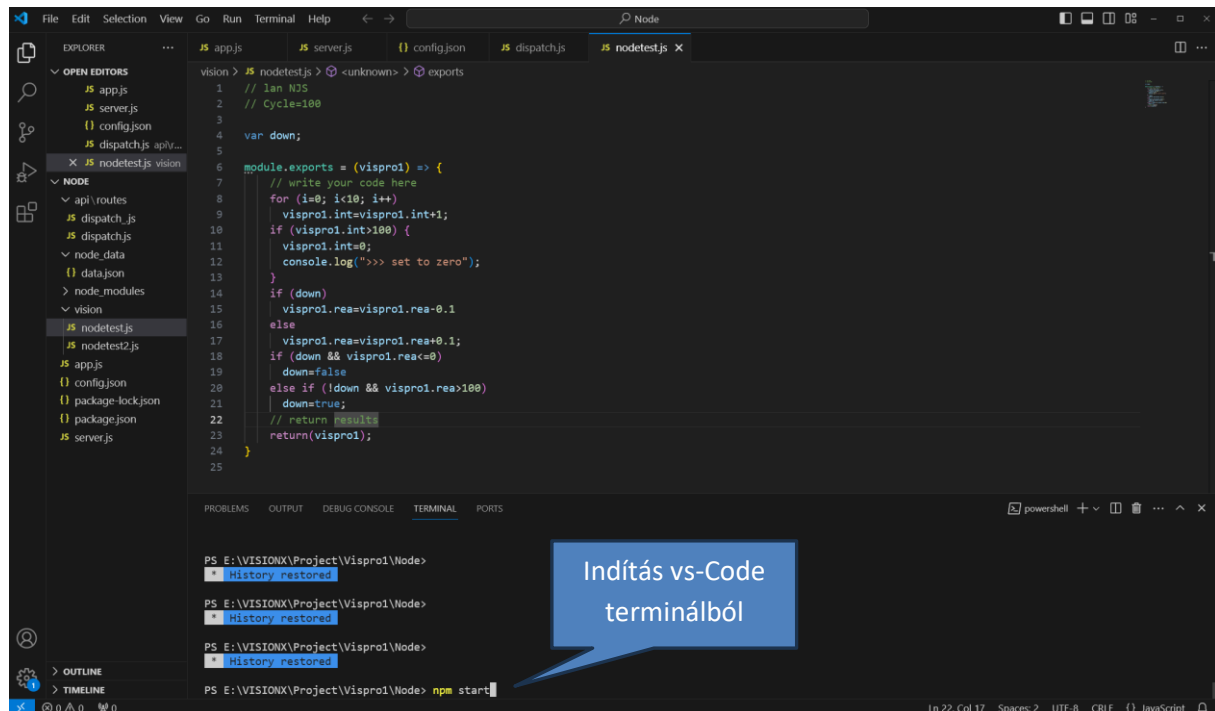
Itt azt láthatjuk, hogy a `">>> set to zero"` szöveg megjelenik időnként a `POST /api...` lekérdezések között. Amiből könnyen kikövetkeztethetjük a kapcsolat módját is: **Rest API**. A VISION a változókat egy POST kéréssel továbbítja a Node felé JSON contentben, a módosításokat pedig a válaszban ugyanúgy JSON-ben kapja vissza.

E művelet időigényét láthatjuk a listában. De **a console terminálban jelenik meg a hiba is, ha valamit sikerül elrontani!** Ezek lennének a Node, mint *rendszer* hibaüzenetei.

Ezen kívül már csak az van, amit mi magunk íratunk ki, hogy ellenőrizzük a program futását. Így válik érthetővé a logging opciók **off** (kikapcsolva), **all** (minden üzenet) és **app** (csak az alkalmazás üzenetei) opciója. De hogyan lehet mindezt menet közben megváltoztatni?

Na erre szolgál a *Node.js config* gomb, amivel a már bemutatott dialógust jeleníthetjük meg újra, hogy abban a terminal üzemmódját megváltoztassuk.

Végül, de nem utolsó sorban, a programot vs-Code alatt is szerkeszthetjük, ami ugyanúgy ingyenes és a VISION-ból is letölthető. Erre szolgál a harmadik gomb:



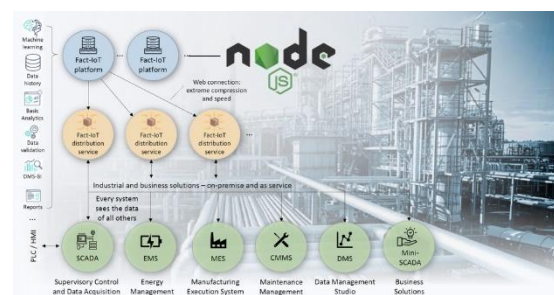
Remek környezet a program futtatására is, megjegyzem. Sőt az egyes függvények belsejébe is betekinthezünk általa, valamint az előre konfigurált Node környezetbe. Akár meg is változtathatjuk, ha kellő jártasságot szereztünk benne. Itt telepíthetünk fel a legegyszerűbben további modulokat (csv-parser, nest+typescript, mongoose, react stb.) is.

Alul a ciklikus program vs-Code-ból történő indításának a módját mutatom (**npm start**). E parancs *nodemon*-t (node monitoring) használva indítja el a programunkat. Ez azt eredményezi, hogy ha bármit módosítunk a kódban, akkor a programot automatikusan újraindítja.

Ezt egyébként maga a VISION is elvégzi, függetlenül a vs-Code-tól és a *nodemon*-tól, ahogy fent már utaltam rá (a futtatás gombot hiába nyomogatjuk, menteni kell).

Magát a node-ot az első ciklikus program indítja el automatikusan, leállítása a VISION alkalmazás bezárásával történik (hacsak másik gépre nem telepítettük a node-ot és a JavaScript környezetet). Végül rögzítsük: **a fent vázolt konfiguráció szervíz környezetben is ugyanúgy fut.**

Sőt az IIoT platform alatt is (ld. jobbra), aminek a bevezetése alapvető szerepet játszott a Node.js környezet kiválasztásában, mint a VISION egyre gyarapodó kódbázisa és a digitalizáció legfontosabb motorja. **A Node.js integrációjával az egész programozói világ vált elérhetővé közvetlenül a VISION-ból.** A ChatGPT néhány feladatára még a kódot is megírja, ha igényled!



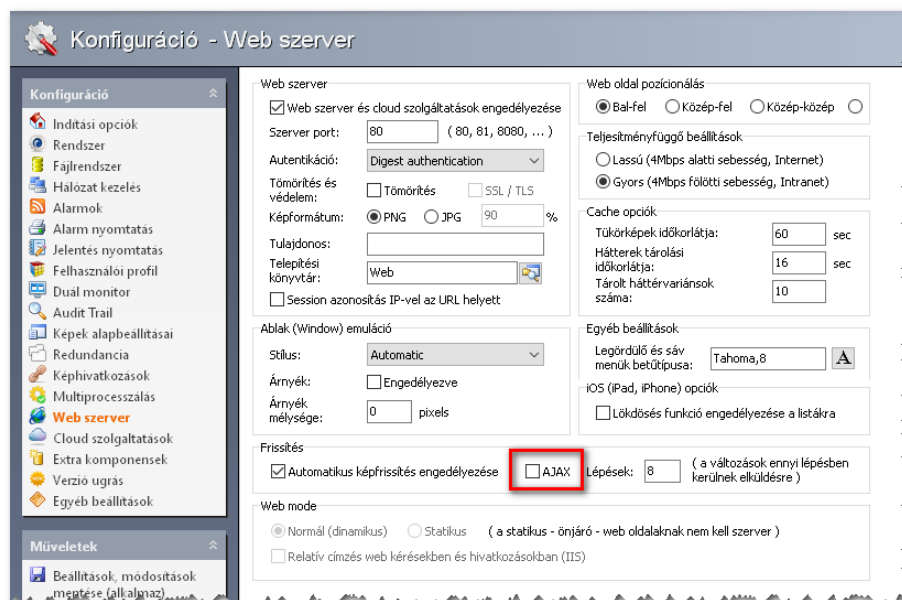
Végül itt olvasható a Node.js dokumentációja: <https://nodejs.org/api/modules.html>

3. Web-es hálózat AJAX használatával

A 4.0 és 4.1. verzióktól egy sokkal gyorsabb és kényelmesebb üzemmódot is beállíthatunk a web-es kliensek frissítésére: AJAX (Asynchronous JavaScript and XML).

Lényegét tekintve arról van szó, hogy a web szerver ilyenkor már nem egy iframe-be küld adatokat – komplett html kódot, benne JavaScripttel –, hanem közvetlenül az ún. *window*-ba, ami VISION DOM html dokumentumát tartalmazza. A legfontosabb különbség azonban az, hogy **aszinkron**, vagyis több párhuzamos kérés is elindulhat egyidejűleg akár. Ha pl. a kezelő lekért a VISION szerverből egy XLSX fájlt, akkor annak előállítása és a letöltése a szerverből eltarthat egy darabig. Ám ha ezalatt a kezelő újabb parancsokat ad ki, pl. még egy fájl leküldését, akkor az is mind végrehajtódik AJAX-ot használva. Az iframe alapú megoldásnál a párhuzamosításról egy javascript programocská próbált gondoskodni (rsx.js) 1500 ms maximális késleltetést állítva be, ami hosszabb kiszolgálási idők esetén a kérés megismétlését igényelhetette. Az AJAX e problémára ad megoldást.

Beállítás a konfiguráció Web szerver dialógusában lehetséges, egy checkbox csupán:



Megfigyelhetjük, hogy amennyiben ezt bekapcsoljuk, a lépcésszám 1-re vált. Ez így helyes. Mivel az átvitel gyorsabb, felesleges több darabban átküldeni, de ettől meg persze magasabbra állíthatjuk ezt az értéket, ha mégis spórolni szeretnénk a web kliens és a szerver közötti tranzakciókkal.

Végül AJAX esetén a leggyorsabb tállózó-oldali frissítést 250 ms-ra csökkentettük, amivel már igen látványos, gyors webalkalmazásokat készíthetünk.

4. Navigálás Ctrl-Click használatával

A modern kódszerkesztőkben ismert megoldás elérhetővé vált a VISION-ben is. Amennyiben a Ctrl gombot nyomva tartjuk, a változókat, típusokat, képeket, szimbólumok neveit stb. a VISION aláhúzza, jelezvén, hogy rákattintva tovább mehetünk a deklarációban. Pl. megnézhetjük így a library-ben írt kódot, a parancs prototípusát, de átválthatunk a kurzor által kijelölt képre vagy szimbólumra is. Ez rendkívül hasznos és kényelmes, mivel megkönnyíti a kódon belüli és a képek közötti navigációt.

X10 Verzió 4.0

1. Addon kiegészítők frissítése

Néhány kiegészítő megváltozott, ezeket érdemes az alkalmazásban is frissíteni a szokásos módon, az Addon menedzserrel. Az érintett kiegészítők a következők:

- AlarmViewSelect (AlarmWindow része)
- TrendImage
- wTrendWindowLight
- wTrendWindowLight2

Az AlarmWindow esetén a web-es működést érinti a változtatás (napló-típus frissítés legördülőből), a TrendImage esetén pedig szépséghibák lettek javítva. Ami azonban a két utóbbi web-es komponensről (wTrendWindow...), ott az adatok web-en történő frissítése változott. Nits Gábor kolléga javaslatára mostantól automatikus a napi-, heti-, havi- és éves nézetek frissítése (tudni kell, hogy a komponens az idő kezdetét és végét használta a html komponens megkülönböztetésére és ezáltal a frissítésére, azonban az aktuális napon pont az nem változik – egészen a nap végéig). Ezt javítottuk. Van ezenkívül még egy apróság:

Felhasználók jelezték, hogy ha az aktuális időszak megjelenítésekor (*Jelen*) belenagyítanak a trendbe, akkor az 1 percen belül visszáll. Ennek pont a frissítés a magyarázata, amire a web-es wTrendWindow esetén percenként kerül sor. Ez kivédhetetlen, mivel a szerver nem tudhatja, hogy a tállzóban milyen lokális állapot uralkodik (nagyítás, kiválasztás). Ezen a problémán úgy segítettünk, hogy hosszabb **vizsgálat előtt meg kell nyomni a még egyszer a *Jelen* gombot**. Ekkor a trendet a program időben 5 perccel visszatolja, ezzel dezaktiválva a jelen állapotot. Ettől kezdve tetszés szerint válogathatunk, nagyíthatunk, exportálhatunk stb., anélkül, hogy a komponens frissülne. Pont az történik, mintha egy nappal korábbi adatokat, tehát a múltat vizsgálnánk, csak most 5 perccel mentünk vissza a múltba. Ez a trükk segíti az kvázi-aktuális trendgörbék elemzését web-en.

2. Új függvények a ConvertDBTable() eljárásban

Az eljárás az [adatbázisok statisztikai elemzése](#) céljából lett bevezetve. Az új függvények a következők:

Integ (változó)	Adat integrálása a tábla elejétől a végéig
Diff (változó)	Adat differenciálása a tábla elejétől a végéig
Ave (változó,factor)	Átlag kiszámítása: ha a <i>factor</i> pozitív, akkor folyamatos átlagképzéssel (pl. abban az esetben, ha a <i>factor</i> = 7, akkor az előző minta 7:1 súllyal van figyelembe véve); ha pedig negatív, akkor annyi környezeti minta átlagát képi minden egyes soron, amekkora a <i>-factor</i> értéke (pl. ha 4, akkor 2 előző és 2 követő mintát átlagolja hozzá az aktuális sorhoz)
PID (változó,P,I,D)	PID algoritmus szerinti számítás a változóra nézve.
PI2D (változó,P,I1,I2,D)	PID algoritmus szerinti számítás a változóra nézve két különböző integráló taggal: az I1 a pozitív irányú változások integráló faktora, az I2 a negatívoké.

Duration(változó) A kiválóan használható időeloszlás függvény számítása a teljes adatsorra.

Példa:

```
ConvertDBTable "TrendSource.xdb", "TrendDestination.xdb",
"summa=telj1+telj2",
"integral=integ(summa)",
"proc=PI2D(summa, 0, 0.11, 0.22, 0)",
"kwpero2=proc/o2",
"O2a=ave(o2, 5)",
"szva=ave(szv, -9)",
"proca=ave(proc, 5)";
```

Hatására a TrendDestination tábla új oszlopokkal (summa, proc, o2a, szva proca stb.) bővül

3. Hálózatok hangolása új konfigurációs paraméterekkel

A VPR.INI fájlba három új tag került bevezetésre, amivel az alaphálózati (1700+ port) kapcsolat-felvételt lehet szabályozni:

```
[Network]
TimeoutListening=300
TimeoutAlive=60
ListeningTrys=20
```

Az első két paraméter másodpercben értendő, az utolsó darabszám. Az összes parameter arra szolgál, hogy a *listen()* állapotot stabilan életben tartsa. Azonban az új op. rendszereken (Windows 10 és szerverek) ez már szükségtelen. Inkább csak bajt okozhatunk vele!

Például az utolsó parameter azt csinálja, hogy az alaphelyzetben 5 percenként (*TimeoutListening*) újraindított *listen()* 20-ik próbálkozása a teljes WSA alrendszer is újraindítja. Valószínűleg szükségtelenül. A *TimeoutAlive* pedig működés közben teszteli az 1700-as bázisportot – megint kikapcsolható.

Érdemes a fenti paramétereket vagy nagyobbra állítani, vagy *en bloc* kikapcsolni az újabb op. rendszereken – egyszerűen 0-ba állítva valamennyit.

4. Beágyazott SY szimbólumok eseménykiszolgálása

Általában a szimbólumon belüli objektumok eseményeit vizsgáljuk. Előfordulhat azonban, hogy magát a teljes SY szimbólumot tekintjük objektumnak, amire érvényesíteni kívánjuk a *Hint*, *Point*, *Killpoint* stb. eseményeket. Ez mostantól IC-be ágyazott szimbólumok esetén is működik.

5. Fact-IoT platform

Erről külön fejezet és dokumentáció készül majd, mivel e téma számos megfontolást és speciális beállítást igényel. Az ipar 4.0 témára és az IIoT kérdésekre külön tanfolyamot szervezünk.

Előzetesen kérlek tekintsétek meg a témáról készült alábbi videót:

https://www.provicon.net/presentation/eneraid/Saia_IIoT_presentation.mp4

Az **IIoT** nem tévesztendő össze az **IoT**-vel. Ez utóbbi egy technológia, ill. internet kapcsolattal rendelkező eszközök hada, az IIoT pedig egy *ipari* platform. Külön elemzést érdemel.

Az IoT eszközök integrálásáról a következő fejezetben lesz szó.

6. IoT eszközök integrálása Home Assistant használatával

Egy újabb jelentős mérföldkövel bővült a VISION szolgáltatások köre: Home Assistant integrálásával. A Home Assistant egy platform, amivel hatalmas mennyiségű IoT eszközt tudunk elérni: gyakorlatilag az összes gyártmányt, ami létezik, tekintettel e platform exponenciális ütemű bővülésére a sok tízezer programozónak és felhasználónak köszönhetően. A Home Assistant használatával kommunikálni tudunk az összes napelem inverterrel, hőszivattyúval, Modbus kommunikáció képes bármilyen eszközzel, kapcsolók, relémodulok garmadájával, IoT érzékelőkkel és vezérlőkkel, kamerákkal és számos további integrációs platformmal, mint a [Google Assistant](#) vagy az [Alexa](#). Ízelítőnek egy lista a Home Assistant-tel elérhető integrációs platformokról és szolgáltatásokról (csak egy töredéke):

amazon alexa Amazon Alexa	ecobee ecobee	ESPHome ESPHome	Google Assistant Google Assistant	Z-Wave Z-Wave	cmus cmus	coinbase Coinbase	ColorExtractor
Google Cast Google Cast	HomeKit Bridge HomeKit Bridge	IKEA TRÅDFRI IKEA TRÅDFRI	KNX KNX	zigbee Zigbee Home Automation	Command Line Command Line	Commonwealth Edison (ComEd) Commonwealth Edison (ComEd)	Compensation Compensation
LEVITON Leviton Z-Wave	LUTRON Lutron Caséta	MQTT MQTT	hue Philips Hue	Bouygues Flexom Bouygues Flexom	Configuration Configuration	Configurator Configurator	Control4 Control4
plex Plex Media Server	Shelly Shelly	SmartThings SmartThings	SONOS Sonos	brother Brother Printer	COOLAUTOMATION CoolMasterNet	Counter Counter	Cover Cover
AccuWeather AccuWeather	acer Acer Projector	Actiontec Actiontec	ADAX Adax	BALBOA water group Balboa Spa Client	CROWNSTONE Crownstone	CUPS CUPS	currencylayer currencylayer
ADGUARD AdGuard Home	BECKHOFF TwinCAT ADS	MyAir Advantage Air	AEMet AEMET OpenData	Baltimore Gas and Electric (BGE) Baltimore Gas and Electric (BGE)	D-Link D-Link Wi-Fi Smart Plugs	Dacia Dacia	DAIKIN Daikin AC
aftership AfterShip	AGENT DVR Agent DVR	Air Quality Air Quality	airQ air-Q	ABB Aurora ABB PowerOne Solar PV	DATADOG Datadog	Date Date	Date/Time Date/Time
aircu Airly	AirNow AirNow	AIRTHINGS Airthings	aws Amazon Web Services (AWS)	belkin we mo Belkin WeMo	De Lijn De Lijn	Phoscon deCONZ	Default Config Default Config
airtouch AirTouch 4	AirVisual AirVisual Cloud	amber Amber Electric	AIRZONE Airzone	blebox BleBox devices	Deluge Deluge	Demo Demo	DENON Denon AVR Network Receivers
AIRZONE Airzone Cloud	ALADDIN CONNECT Aladdin Connect	Alarm Control Panel Alarm Control Panel	AlarmDecoder AlarmDecoder	ambi CLIMATE AmbiClimate	DENON Denon Network Receivers	fx Derivative	Deutscher Wetterdienst (DWD) Weather...
Alert Alert	ALPHA VANTAGE Alpha Vantage	amazon alexa Amazon Alexa	amazon alexa Amazon Alexa Custom Skill	HAYWARD AquaLogic	Device Tracker Device Tracker	devolo Home Control devolo Home Control	devolo Home Network devolo Home Network
amazon alexa Amazon Alexa Flash Briefing	Ambient Radio Weather Network Ambient Radio Weather Network	fire tv Amazon Fire TV	Amazon Polly Amazon Polly	ARRIS Arris TG2492LG	DHCP Discovery DHCP Discovery	Diagnostics Diagnostics	Dialogflow Dialogflow

Jelenleg [2524 platform](#) illeszthető! Bár a Home Assistant elnevezés arra utal, hogy ezzel otthoni okoseszközöket tudunk kezelni, ennél valójában jóval többről van szó, mivel ipari alkalmazásokban is egyre gyakrabban találkozhatunk a megoldással (IoT mérőhálózatok). De gondoljunk csak arra, hogy egy egyszerű vezérlést kell megvalósítanunk (pl. napelem inverterrel), vagy kamera képeket szeretnénk megjeleníteni a VISION-ben. Egy modern SCADA nem nélkülözheti az IoT eszközök kezelését.

Amivel el is jutottunk az alapkérdéshez: hogy jön mindez össze a VISION-nel? Hogyan lesznek elérhetők a VISION-ben azok az IoT eszközök és szolgáltatások, amelyek a Home Assistant-ben látszanak?

Ehhez mindenképp először azt kell tisztázni, hogy a Home Assistant önálló szerveren fut, Linux alatt, **lokálisan**. Ez lehet klasszikus szerver, sőt miniPC is, de pont az a lényeg, hogy még drága eszköz sem szükséges hozzá: elfut jóval olcsóbb hardwareken, mint a [Raspberry Pi](#), [Odroid-N2](#), a Home Assistant [Yellow](#) vagy [Blue](#). Maga a platform ingyenes, sőt a rá telepíthető integrációs szolgáltatások is mind – csak a HW-ért kell fizetni néhányszor 10eFt-ot. A telepítése nagyon egyszerű, erről külön videót fogok csinálni. Ahogy a legfontosabb kezdeti beállításokról és az alapvető eszközök integrálásáról is.

A rendszer támogatja a szokásos IT kommunikációs médiumokat: vonalas Ethernet, Wifi, Zigbee, Z-Wave, Bluetooth – hogy csak a leggyakoribbakat említsem. De van USB és soros vonal is.

Kapcsolat a VISION-nel

Annak érdekében, hogy a Home Assistant-et összekapcsoljuk, engedélyezni kell a Home Assistant-ben a [HTTP szolgáltatást](#). Ez teszi elérhetővé az eszközt a [Rest API](#) interfészén. Innen kezdve már csak pár kattintás és minden adat bent van a VISION-ben – a historikus adatok is! –, mi több, tudunk parancsokat kiadni, vezérelni és kameraképeket is meg tudunk jeleníteni a VISION-ben. Nézzük, hogyan!

Konfiguráció

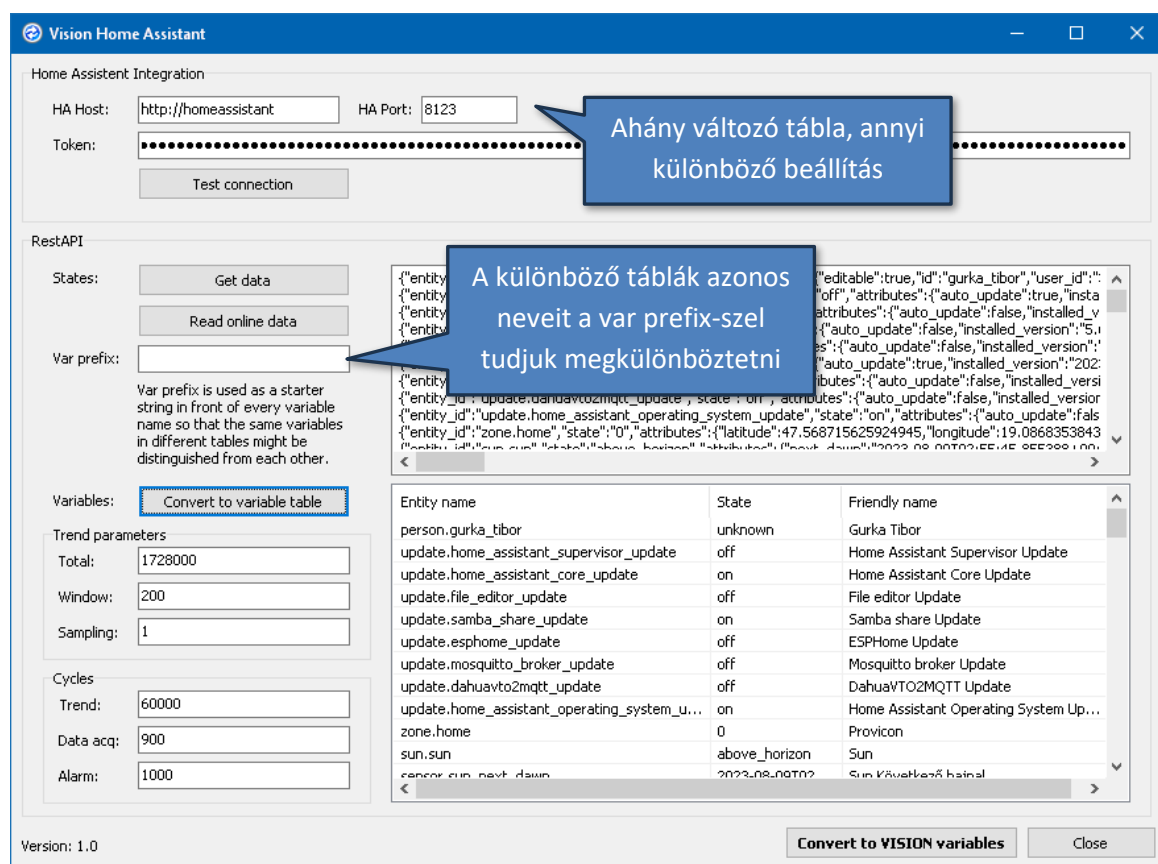
Két taszkot kell bekonfigurálnunk: egy változó táblát és egy kommunikációs taszkot.

1. Új változók létrehozása

Mint látható, van itt egy új legördülő menünk az *Integráció forrása* szöveg mellett és abban ott van a Home Assistant. Ezt kell kiválasztani.

Egy kis segédprogram fog elindulni, ami az egész változó táblát legyártja nekünk, benne a Home Assistant-ben található összes entitással. Később a VISION táblák szerkesztésénél található frissítés gombbal tudjuk felvenni, átnevezni, törölni a változókat az aktuális állapotnak megfelelően.

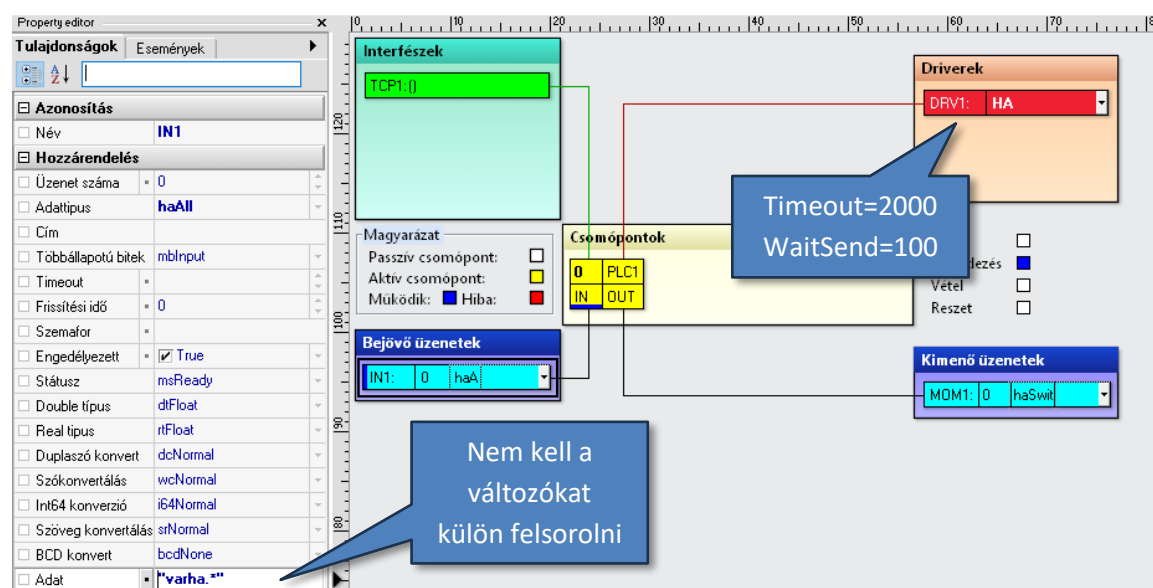
A következő képen látható a segédprogram (*VisionAssistant*). Itt meg kell adnunk egy URL-t, amin keresztül a Home Assistant szerver látszik – ez lehet HTTPS –, a portot és a biztonsági tokent, amit a Home Assistant-ben hozunk létre. Ez biztosítja, hogy csak mi tudunk hozzáférni a rendszerhez. Miután ezeket a paramétereket megadtuk, a VISION letárolja és többé nem kell törödnünk vele – sem a változók frissítésekor, sem a kommunikáció során.



A **Convert to VISION variables** gombot kell megnyomni. Előtte állítsuk be a trend paramétereket és a ciklusidőket – később a program az itt megadott preferenciáinkat fogja felkínálni.

Mint a fenti képen látható, a program lekérdezte az állapotokat JSON kódban, alatta pedig egy táblázat látható a változók nevével, értékével és leírásával (természetesen a VISION változó tábla ennél bővebb). Ezzel a változókat előállítottuk.

2. Új kommunikáció létrehozása



Az új driver neve: **HA**. A TCP interfésznél semmit nem kell megadni és a be- kimenő üzeneteknél sem kell felvenni semmilyen változót. Elég csak az elérés típusát megadni (*haAll*, *haSwitch* vagy *haLight*), valamint adatnak annyit, hogy *varha.**, feltéve, hogy **varha.val** néven hoztuk létre a változó táblát. Így biztosítjuk, hogy a VISION a *varha*-ban lévő összes változóhoz hozzáférjen input és output üzenetekben. Egyetlen dolgot kell csak eldöntenünk, hogy ti. az összes változót engedélyezzük-e kimenetre is, vagy csak a kapcsolókat (*haSwitch*). Ez utóbbi ugyanis eléggé tipikus.

Tanácsnak még annyit, hogy ne kommunikáljunk felesleges gyorsan az eszközzel, abban ui. az adatok nem frissülnek olyan villámgyorsan és a processzor terhelés növelésével az csak feleslegesen melegszi. 100-1000 ms a javasolt beállítás, amit a legegyszerűbben a *WaitSend*-del hangolhatunk.

Vegyük észre, hogy a kimenet MSOM (változás orientált). Ez fontos, különben a rendszer nem fog megfelelően működni. Parancsot emiatt akkor fog küldeni a VISION a Home Assistant-nek, ha módosítást hajtunk végre a változókon. **Alapból a *light*. és *switch*. kezdetű változónevek működtethetők úgy, hogy device szinten is végrehajthatódnak, a többi parancs csak a HA belső reprezentációs értékeit írja felül.** Ez egy Home Assistant specialitás. Például egy bemenet állapotát is felülírhatjuk. Viszont a rendszer további valódi parancsokkal és szolgáltatásokkal bővíthető a HA oldalán Python programozással. De programozható YAML konfigurációs nyelven is – ezt ők automatizációnak hívják. PLC szintű műveleteket is beprogramozhatunk tehát és a HA-t, mint PLC-t használhatjuk. Ezekkel külön dokumentációban foglalkozok majd, ami kifejezetten a HA oldali lehetőségekről szól.

Megjelenítés

A változók használatával és megjelenítésével kapcsolatban ugyanazok a szabályok érvényesek, mint általában is. A kameraképek és a hisztorikus adatok megjelenítését azonban érdemes részletezni.

1. Kameraképek

Mivel a Home Assistant képes kommunikálni a világ valamennyi (okos, új) kamerájával, mostantól ezek mind megjeleníthetők a VISION-ben is. A modern eszközök már **mesterséges intelligencia szolgáltatást is nyújtanak**: alak-, állat- és járműfelismerést, természetesen mozgásérzékelést, de az okosabbak fel tudnak ismerni otthagyt tárgyakat, sőt rendszámot is. Ezek, mint változók, mind látszanak a VISION-ben.

Ha vannak a rendszerben kameráink, a változólistában megtalálhatjuk a kamera-állapotot reprezentáló *camera*. kezdetű változókat. Ezekből arra a diszkrét változóra, aminek jellemzően *idle* és *motion* a két állapota (TCamera a típusa), ki kell adni egy módosítás parancsot (*idle* ⇒ *motion*):

```
Discretetes
camera.pl346_0: CAMERA, @TCamera
binary_sensor.cameral_motion: SENSOR, @TOnOff
binary_sensor.cameral_person: SENSOR, @TOnOff
binary_sensor.cameral_vehicle: SENSOR, @TOnOff
binary_sensor.cameral_pet: SENSOR, @TOnOff
camera.cameral_sub: CAMERA, @TCamera
light.cameral_floodlight: LIGHT, @TOnOff
light.cameral_infra_red_lights_in_night_mod: LIGHT, @TOnOff
```

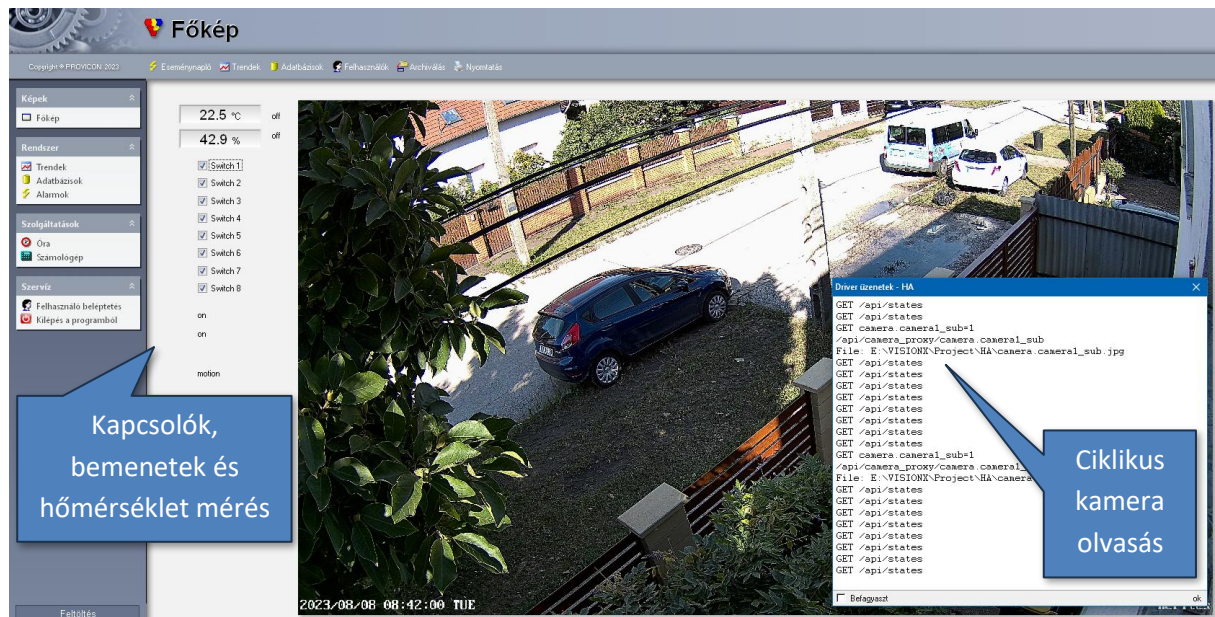
TAttr begin
idle:
motion:
End

H75ZW1N3NV39Q65MXGW27ED
ion | 01H75QZPXQNR6M601JBTM7M7AH
son | 01H75ZDWOK8NR3VP1V60WJJS8T
idle | 01H75SMT5HJJ2SS3RV2PJAV8HK
01H74G28NN3ENQW67XOTD4Q1TY
01H75ZW1N4WXP03DQV4Z6E7SS
'Kerites Floodlight | 01H74G28NQMS045GWSNH1218H6
'Kerites Infra red lights in night mode | 01H74G2

Például a *Refresh* ágbán, így (mivel a változó jellemzően 0-ban van):

```
camera.cameral_sub = 1; 'set to motion
```

Ennek hatására a kommunikációs driver ugyanilyen néven legyárt egy .jpg fájlt az alkalmazás alatt, ami egy *snapshot*. Ezt kell feltennünk a képre **kikapcsolt cache mellett** (LD). Majd a változót, amíg a képen vagyunk, monjuk 500 miliszekundumonként változtatni – ciklikusan. Sajnos sokkal gyorsabban nem lehet, mert a kép legyártásának az eszköz oldalon van némi időigénye (pláne 4K képek esetén):



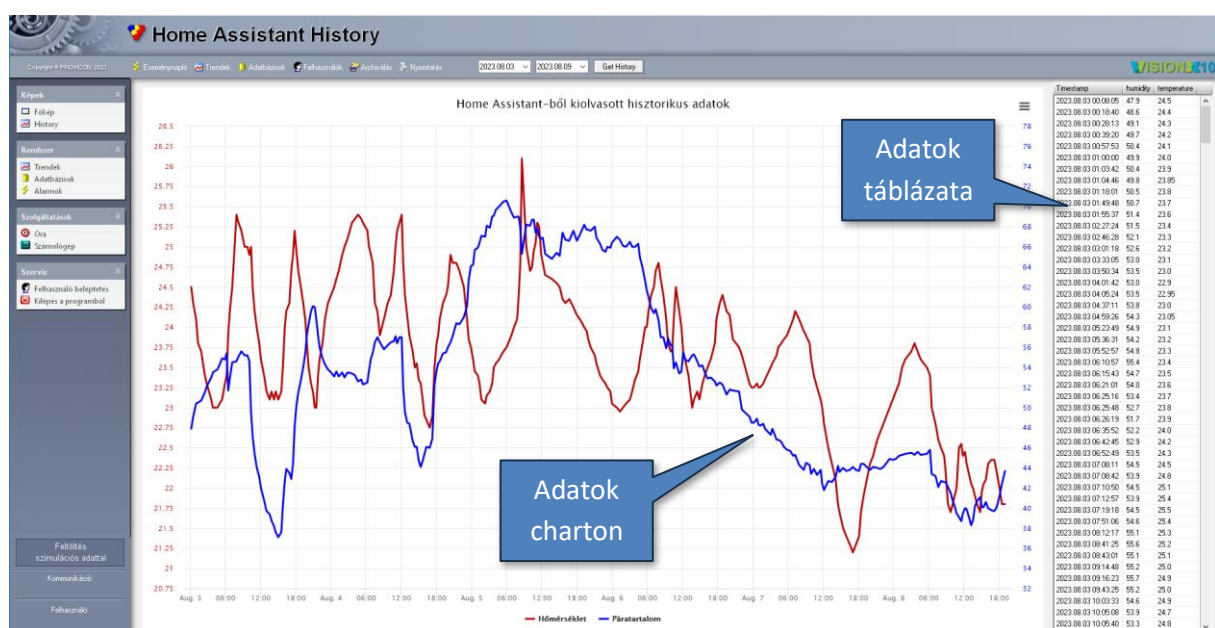
A felvétel készítésre is képes rendszereknek még az archív felvételeit is le tudjuk kérdezni.

2. Hístorikus adatok

A Home Assistant képes tárolni is az adatokat a saját szerverén, amit azután bármikor lekérdezhetünk belőle és letehetjük egy XDB adatbázisba (onnan meg átkonvertálhatjuk trendbe, vagy más SQL adatbázisba). Az ehhez szükséges új parancs neve *haHistory*, definíciója a következő:

```
haHistory:: procedure (datatable: string; timefrom: long; timeto: long; vars: list of pvar)
```

Az első argumentum a létrehozandó adatbázis, majd az idő kezdete és vége, végül a változók listája. A program az időbélyeg alapján próbálja relációba rendezni a különböző vektorokat, hogy egy táblázatban / charton is ábrázolhatók legyenek. Tudni kell, hogy az IoT eszközök időbélyege egyáltalán nem ekvidisztáns, mivel az eszköz jellemzően csak akkor küld adatot, ha az változik. Az egyeztetés során a program köztes pontokat interpolál. Az eredmény látható a következő ábrán:



3. Térkép objektumok

A Home Assistant kezel ún. *tracking objektumokat* is, amit *zone*. változókhöz (entitás) rendel. Ezek lehetnek statikusak és dinamikusak is. A tipikus zóna kezelés 3 változóból áll VISION szinten: egy lélekszámlálóból (hány ember tartózkodik a zónában) és a zóna GPS koordinátáiból. Kézenfekvő, hogy ezeket az objektumokat térképen is meg tudjuk jeleníteni, ahogy a Home Assistant maga is. Mivel azonban a VISION képes dinamikus objektum követésre, így a tracking objektumok is folyamatosan mozoghatnak a VISION térképeken és mutathatják a hozzá tartozó jellemzőket:

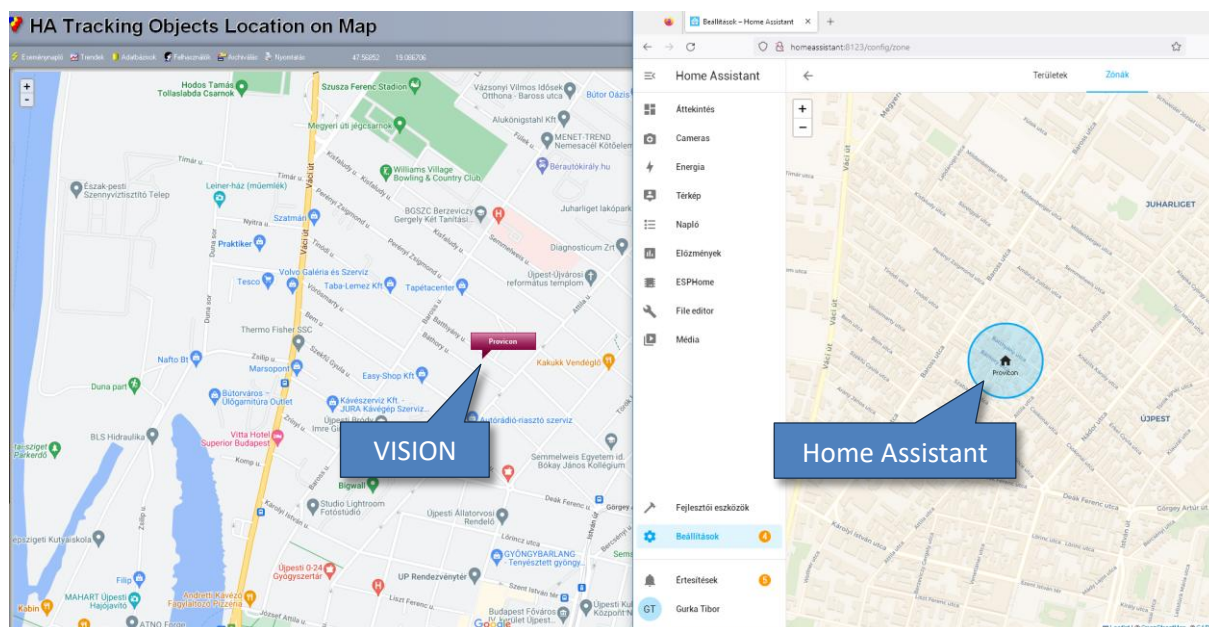


Figure 1 Folyamatos objektum követés Google tréképen

Ahogy arrébb mozgatjuk az objektumot a Home Assistant-ben, vagy mozog magától a *tracking device* által küldött GPS koordinátát követve, úgy változik – akár pár milliszekundumos gyakorisággal – az objektum pozíciója és minden hozzárendelt adata (mérések, zóna-állapot stb.) a VISION-ben.

A megoldást jellemzően jármű- vagy járórkövetésre használják, ill. a mozgásvezérlőkkel együtt az épület különféle zónáiban található munkatársak / ügyfelek számának és mozgásának a nyilván-tartására, követésére.

4. Speciális entitások

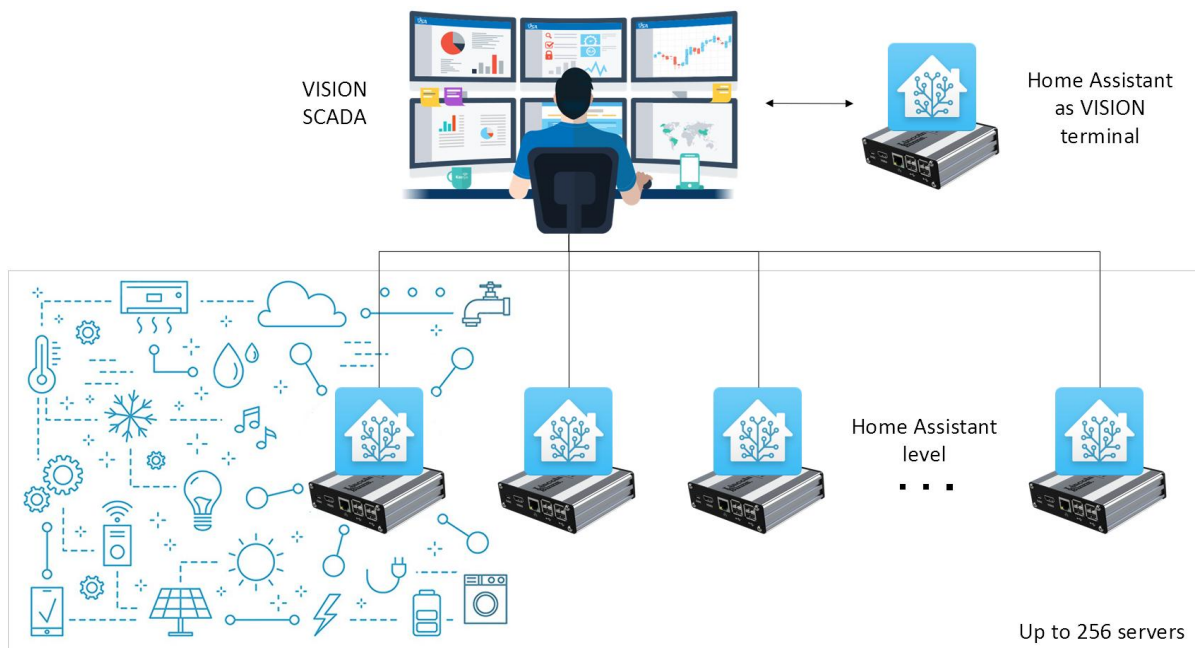
Számos további változót találhatunk a Home Assistant-ben, amik klasszikus folyamatmegjelenítő alkalmazásokban nem szokásosak. Ezeket is kezeljük. Ilyenek a különféle állapotmódosító listák (pl. TV-csatorna), az időjárás jelentés, időjárás előrejelzés, napfelkelte, naplemente időpontja, eszközök update állapotai, sőt még az update folyamat maga is mind kezelhető VISION-ből.

A lényeg, hogy a Home Assistant minden adattípusát, az összes *entitást* hozzárendeltük változókhöz és kezeljük azok tulajdonságának és használatának megfelelően és a megjelenítésük is ezt követi.

Home Assistant integráció

Többször utaltam rá, most külön is kihangsúlyoznám: nem csak 1db HA szerver lehet a rendszerben, de akár több száz. A VISION ilyenkor mint fölérendelt HA integrációs platform funkcionál, amely képes párhuzamosan akárhány HA szerverrel kommunikálni (mintha csak PLC lenne). Annyi csak a nem túl szigorú megkötés, hogy ezek adatai eltérő változó táblákba kerülnek és akként frissülnek. Például, amikor historikus adatokat olvasunk ki a HA-ból, azok származhatnak eltérő HA szerverek-

ből is (mivel a változólista elemei több változótablát is érinthetnek). Közben a VISION kommunikál a többi ipari vezérrel, PLC-vel és RTU-val. A rendszer adatai tehát végül a VISION-ben jönnek össze. Így alakul ki a Home Assistant integrációval bővített teljes, elosztott ipari mérőhálózat.



De ez nem feltétlenül köti meg az alá-fölérendeltségi viszonyokat a VISION javára. Mivel a VISION is lehet forrása a HA-nak. Ugyanis maga a VISION is képes létrehozni abban entitásokat – ez úgy történik, hogy bizonyos szabályokat követve a HA-hoz rendelt változótablában új változókat veszünk fel –, amiből azután a HA szintjén tudunk dashboardot rajzolni, diagramot, számkijelzőt, kapcsolót stb. A HA mint egy Linuxos VISION terminál funkcionál tehát, a saját reszponzív, websocket vezérelt, gyors web-es felületét használva megjelenítésre.

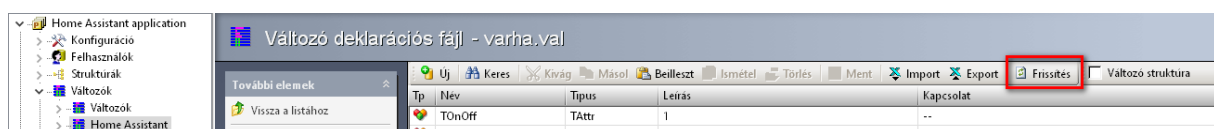
Automatizáció VISION-ből Home Assistant-be

De még ennél is többről van szó, ami a két rendszer összekapcsolását illeti. Ugyanis a VISION-ben készített vezérlők egy részét le tudjuk tölteni a Home Assistant-be, mint automatizációt (YAML kódban). Így az végül, bár VISION-ben készült, Linux alatt fut.

Attól függően, hogy e megoldás milyen visszhangra talál majd az ügyfelek körében, további vezérlőket is implementálni fogunk a jövőben és VISION-ben rajzolt dashboardokat letölteni, megjeleníteni a Home Assistant grafikus felületén (*lovelace*). Tehát, hogy ne csak mint adatforrás funkcionáljon a VISION a Home Assistant irányába, de mint egy alternatív HA konfigurációs platform is használható legyen...

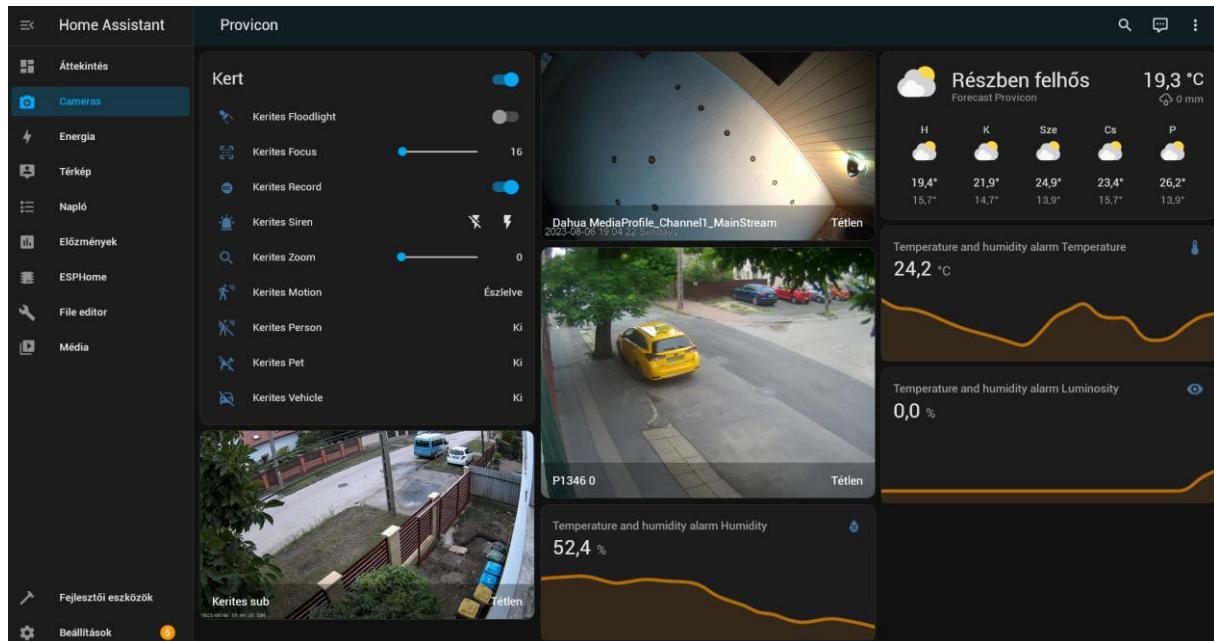
Adatok frissítése

Felvetődik a kérdés, hogy mi történik, ha bizonyos entitásokat törölünk, vagy átnevezünk, ill. újakat hozunk létre a Home Assistant oldalán? Hogyan kerülnek ezek a változások át a VISION-be? Így:



Extra szolgáltatások

Magának a Home Assistant-nek is van saját web-es menedzsment és megjelenítési felülete, amin számos dashboard elem közül válogathatunk és azt rajzolunk rá, amit csak akarunk:



Az érdekesebb szolgáltatásokhoz tartozik, hogy **tud a rendszer beszélni** is (TTS), sőt a *Google Assistant* és / vagy az *Alexa* integrálásával **beszédvezérlésre** is képes. Gyakorlatilag bármit hozzáintegrálhatunk a rendszerhez, ami az IT világban előfordul. A fenti dashboard képen **időjárás előrejelzést** is mutatok, ami természetesen a VISION-ben is láttatható ugyanígy (vagy okosabban)! A kamera pedig képes közvetíteni a kép mellett az audio csatornát.

A Home Assistant különleges tulajdonsága, hogy **a hozzákapcsolt eszközöket automatikusan felismeri**, például a kamerákat is, amit a hálózaton keresztül elér. De felismeri a napelem invertert, sőt még az okos televíziót is (aminek aztán képes váltani a csatornáját, kezelni a hangerőt stb.), az erősítőt, az SNMP eszközöket (pl. nyomtatót), lehet vele üzenetet küldeni okos telefonra, MQTT üzeneteket küldhet a VISION-nek és más integrációs platformoknak, sorolhatnám a végtelenségig.

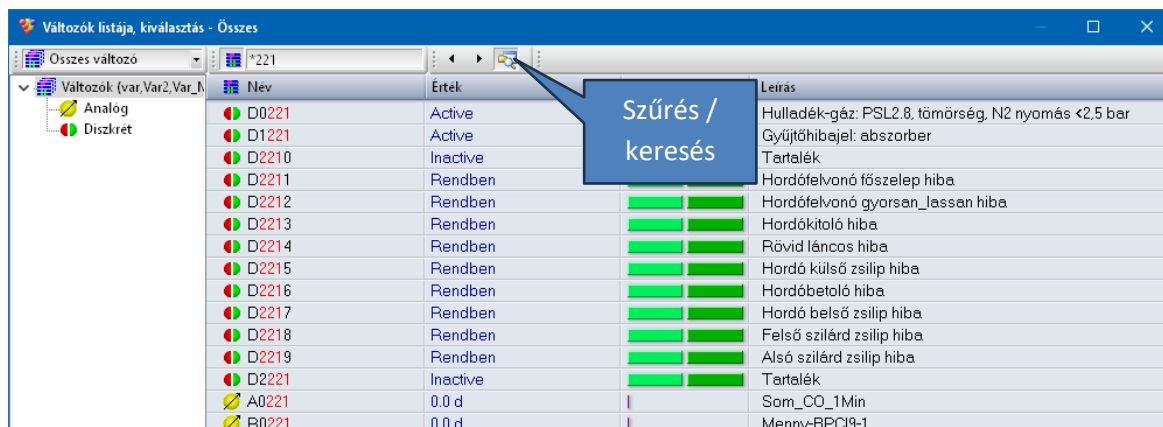
A rendszer sokrétűségét a legplasztikusabban talán a [Tuya platformon](#) keresztül lehet bemutatni. Ez a több ezer felhőszolgáltatás közül az egyik. Hőmérséklet, páratartalom és fénymérők, mozgás-érzékelők, villamos-, gáz- és vízfogyasztás mérők, biztonsági eszközök, világítás, relé modulok 10 dollárért, kapcsolók, nyomógombok, ajtócsengő és még nagyon sok minden érhető el ezen a platformon (is). Az eszközök nem a Home Assistant-tel kommunikálnak, hanem a "felhővel", a Tuya globális szervereivel, amelyek megtalálhatók minden kontinensen. A szenzoroknak valamilyen módon internet elérést kell biztosítani. Onnan kezdve a Home Assistant már a felhőből hozza el az adatokat, normál internet kapcsolaton. A lényeg, hogy nincs szükség vezetékezésre.

Ha mindezt nem akarjuk, lokális kommunikációra képes eszközt is találunk bőven. Például 25 eFt-ért 8 ki/be relé modul illeszthető a rendszerhez, ami USB-n, Etherneten, Wifi-n keresztül is képes lokálisan kapcsolódni (tehát a felhőt kikerülve). Sőt van olyan kombinált [ESPHome](#) eszköz, ami 16 ki- és bemenettel, plusz 8 analóg bemenettel rendelkezik egyetlen kártyán – és még programozható is. Hihetetlen a választék a lokális kommunikációra képes eszközök között is.

X10 Verzió 3.5

1. Változókat érintő változtatások

- a) A változók keresését és szűrését könnyíti meg a '*' karakter használata: amennyiben a szűrő / kereső mező csillag karakterrel kezdődik, a mögöttes szövegrészt a változón belül bárhol elfogadja; ha a keresés / szűrés nem csillaggal kezdődik, találatnak a töredékszó változó elejétől vett egyezése számít, ahogy eddig:



Lényegében arról van szó, hogy a kereső / szűrő mostantól értelmezi a * *wildcard* karaktert, ahogy az reguláris kereső-kifejezésekben megszokott.

- b) A találatot a program pirosra színezéssel jelzi (ld. fenti képet)
- c) A változók maximális hosszát megnöveltük 52 karakterre (44 volt).

2. Trendeket érintő változtatások

Az SQL trendek TMP könyvtárban található logja (SQL.log) további bejegyzésekkel bővült, segítő a hibakeresést.

A cache méret a korábbi 200%-ról 300%-ra nőtt. Így optimális esetben az átmeneti tárnak csak az 1/3-át használja ki a program, mivel ekkor menti a memória tartalmát az SQL adatbázisba. A nagyobb cache mellett az szólt, hogy nagyobb tartalékokat képezzen; az optimális méret akár kétszeresét is képes legyen tárolni, ha a program esetleg nem tudná időben kimenteni a cache tartalmát, mert egyéb feladatokkal van elfoglalva. A 100%-ot is kisebb valószínűséggel éri el és az sem okoz adatvesztést!

A megváltozott cache méret a trend-diagnosztikában is látszik. Mostantól a piros sávnak nem 50%, hanem 33% körül kell változnia.

3. Konfigurálható többmonitoros megjelenítés

Igaz, hogy csak kézzel állítható, de a projekt alatt található PRJ.INI fájlban ezentúl minden egyes monitor pixel-eltolási pozícióját be tudjuk konfigurálni:

```
[MultiMonitor]
Default=0
wx1=0
wy1=-1080
wx2=2560
wy2=-500
wx3=2560
wy3=550
```

Először Default=1 opciót kell kikapcsolni (Default=0) és azután a plusz monitorok pozícióját a 3 plusz monitoron külön-külön beállíthatjuk. Az eltolás a főmonitorhoz képest értendő pixel koordinátában.

4. Kisebb hibajavítások, változtatások

- Login: ha a felhasználót úgy vettük fel, hogy az első bejelentkezéskor jelszómódosítást kérjen, akkor az kliens oldalon csak SQL user adatbázis használatával működött, binárisal nem. Ezt javítottuk.
- Login: Némileg változott és letisztult a felhasználók felvétele dialógus.
- Több ablakos nézet: X10-es környezetben hibás volt a második, harmadik stb. monitorra kitett képen található template megjelenítése.

Ehhez kapcsolódóan készült egy új függvény: *ActiveMonitor*, ami azt mondja meg a program számára, hogy épp melyik monitoron fut. Ez felhasználható például a template menüpontjainak a színezésére egy tömbváltozó indexelésén keresztül:

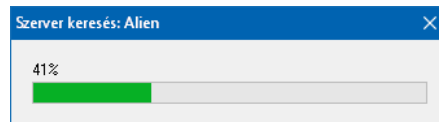
```
BX3:BX " Folyamatábra", 18.5, 5
BEGIN
    Position= 2.875, WindowHeight-40.25
    TxBkgndColor= CVALUE (ImageNumber[ActiveMonitor]=5, rgb(255, 227, 129), MenuColor)
    Font= "Arial", 10, Bold, 0
    Glyph= "Icon\16x16\Cog3.bmp"
    GlyphOffsetX= 4
    Menustyle= bsGradient
    Enable= ImageNumber[ActiveMonitor] <> 5
    Visible= ImageNumber[ActiveMonitor]=2 OR ImageNumber[ActiveMonitor]>=5
    Cursor= crPoint
    Anchor= akLeft+akTop
    Click exec "Techfolly2"
END
```

Itt az *ActiveMonitor* az *ImageNumber[]* tömböt indexeli, amiben azt tároljuk el, hogy épp melyik képen járunk. A különböző monitorok különböző képeket mutatnak, de a template mindig a megfelelő menüpontot színezi el az eltérő képeket megjelenítő monitorokon.

X10 Verzió 3.4

1. Kapcsolódás redundáns szerverhez

Redundáns környezetben legalább 2 szerverünk van. Abban az esetben, ha mindkét szerver elérhető, a rendszer gyorsan, szinte észrevehetetlenül kapcsolódik. De ha a kapcsolódási folyamat tovább tart fél másodpercnél – például azért, mert a második szerver nem elérhető –, akkor megjelenik egy progress bár:



Alaphelyzetben ez 5 másodpercig látható (100% = 5 sec), mielőtt a kapcsolatteremtést a rendszer feladja. A rendszerek sokfélesége azonban indokolta a kapcsolódási idő konfigurálhatóvá tételét. A default 5 másodperc mostantól a VPR.INI fájlban módosítható (System szekció):

TimeFindServer=5 // timeout másodpercben

Ez az első fontos módosítás. A másik változtatás, hogy ha bezárjuk ezt az ablakot, akkor azonnal befejezi a szerver keresését. Korábban ezt nem lehetett megtenni, mivel a program az ablakot azonnal visszatette. E módosítással a szuperprojekt környezetben futtatott alkalmazások bosszantóan hosszú betöltési folyamata rövidíthető le, ha csak egy szerverünk van (tesztkörnyezet), mivel nem kell feltétlenül végigvárunk a projektenkénti 5+ másodperces keresést.

A harmadik változtatás a szükséges fájlok automatikus replikációja redundáns környezetben (pl. a DATA.TRN trend adatfájl lemásolása az aktív szerverből a standby szerverbe, hogy szerver átvétel esetén ott folytassa a program a trendelést, ahol az előző szerver abbahagyta). Az adatmásolást korábban a *replikáció* alatt kellett beállítani, de ez szinte az összes gyakorlati esetben problémát okozott a felhasználóknak (fájlnév konvenciók, XDB adatbázisok...) Ezért ez mostantól automatikus.

Megjegyzendő, hogy a **javasolt beállítás továbbra is az SQL adatbázis használata az összes adattípusra**: trend, alarm, adatgyűjtés, audit trail, felhasználó kezelés. SQL adatbázisok esetén a **szerverváltás nem igényli az adatbázis cserét**, ezért adatmásolásra sincs szükség, hisz mindkét szerver ugyanazzal a *ConnectionString*-gel kapcsolódik, az adatbázis redundanciájáról pedig a *Failover cluster* beállítások gondoskodnak.

A változtatások összefoglalva a következők:

- Konfigurálható szerver keresési idő (*TimeFindServer*)
- Szerver keresés abortálható ablakbezárással
- Automatikus adatmásolás nem SQL adatbázis komponensek esetén

2. Kisebb hibajavítások

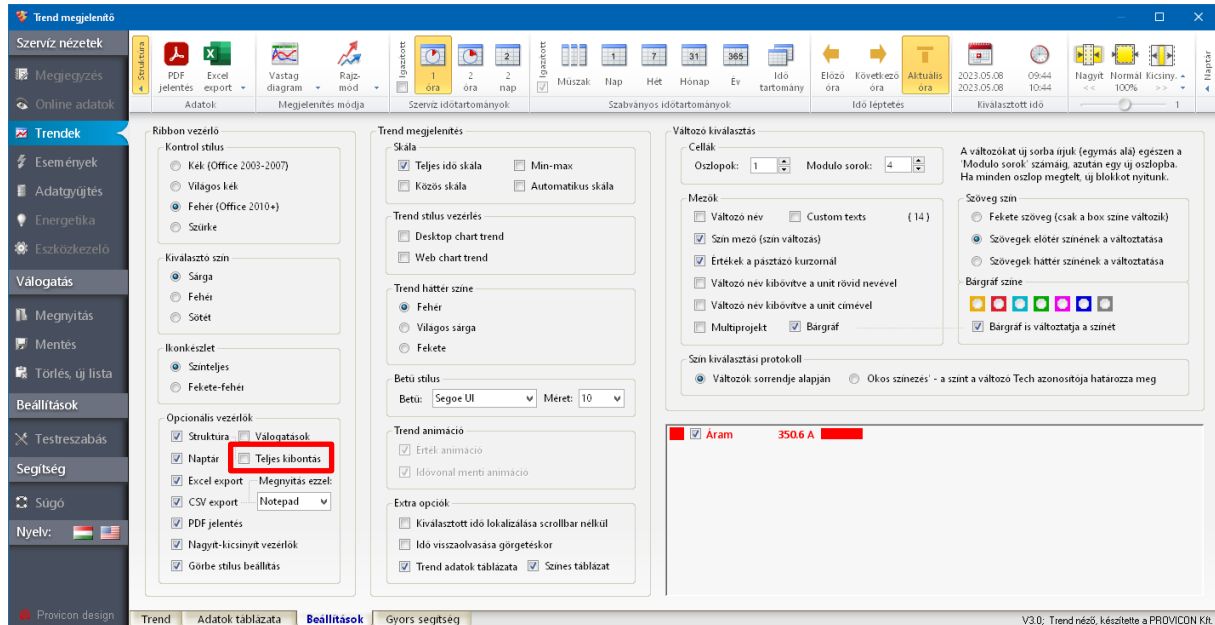
A programban a következő javítások történtek:

1. Alarm statisztika és alarm térkép megjelenítése, ha a naplózásba az operátor nevét is bele vesszük
2. Alarm, kommunikáció és vezérlők megjelenítési hibája "különleges" felbontású ablakok közötti váltás esetén
3. PostgreSQL adatbázisba konvertált felhasználó adatbázis esetén voltak hibák

X10 Verzió 3.3

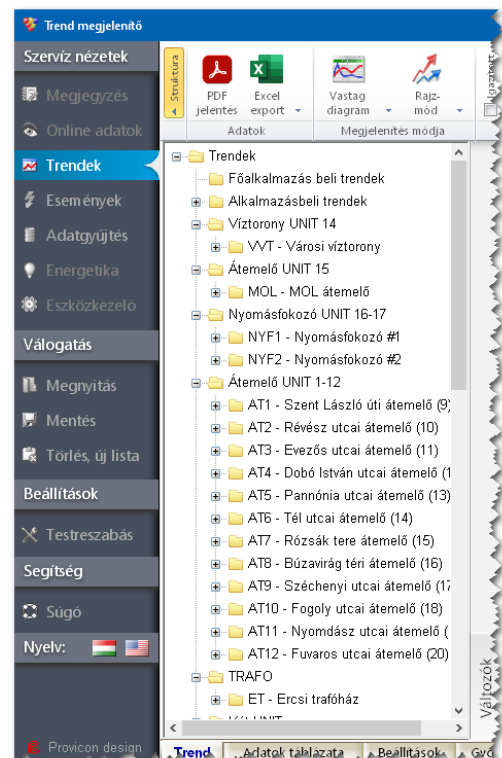
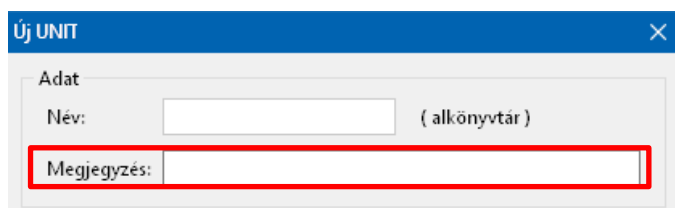
1. Unit-struktúrák finomítása

Az *AlarmWindow*, *TrendWindow* és *DataWindow* kiegészítőkből 3.2 verziótól kezdve megjelenik a unitok struktúrája is (ld. előző pontot). Ezen a téren további finomításokra került sor: a unitcsoportok külön ág alá kerültek és cseréltük az ikonokat, hogy egységes legyen a megjelenés. Választhatunk továbbá *teljes kibontású* fastruktúra és az *első szintig kibontott* struktúra között:



E módosításra a sok unitos rendszerek hosszúra nyúlt fastruktúrája miatt volt szükség. Az új beállító *checkbox* miatt érdemes frissíteni az érintett kiegészítőket. Tudni kell továbbá, hogy a **kibontási mód változtatása csak újraindítás után érvényes**, mivel a struktúrát a program a memóriában tárolja el (cache).

Az első szintig kibontott fastruktúra sokkal áttekinthetőbb, az egyes alrendszereket is könnyebb így lokalizálni. A jobb oldali képen látható az is, miért érdemes megjeleníteni a unitcsoportokat. Ezzel kapcsolatban tudni kell, hogy a unitcsoportokhoz rendelhetünk beszédes leírást, ahogy az ábrán is látszik. Ez a vpr-ben a komment. Amikor új unitobjektumot veszünk fel a programmal, a dialógus felkínál egy megjegyzés mezőt, de azt sokan nem töltik ki:



Viszont utólag is könnyen beírhatjuk a kommentet szövegesen a vpr fájlban:

```

Units
VIZTORONY: begin 'Viztorony UNIT 14
               VVT: "VVT" 'Városi viztorony
             end
MOLATEM: begin 'Átemelő UNIT 15
             MOL: "MOL átemelő" 'MOL átemelő
           end
NYOMFOK: begin 'Nyomásfokozó UNIT 16-17
             NYF1: "Nyomf #1" 'Nyomásfokozó #1
             NYF2: "Nyomf #2" 'Nyomásfokozó #2
           end
ATEMELO: begin 'Átemelő UNIT 1-12
             AT1: "Szt.László út" 'Szent László úti átemelő (9)
             AT2: "Révész u." 'Révész utcai átemelő (10)
             AT3: "Evezős u." 'Evezős utcai átemelő (11)
             AT4: "Dobó I. u." 'Dobó István utcai átemelő (12)
             AT5: "Pannónia u." 'Pannónia utcai átemelő (13)
             AT6: "Tél u." 'Tél utcai átemelő (14)
             AT7: "Rózsák tere" 'Rózsák tere átemelő (15)
             AT8: "Búzavirág tér" 'Búzavirág téri átemelő (16)
             AT9: "Széchenyi u." 'Széchenyi utcai átemelő (17)
             AT10: 'Fogoly utcai átemelő (18)
             AT11: 'Nyomdász utcai átemelő (19)
             AT12: 'Füvaros utcai átemelő (20)
           end
TRAF0: begin
             ET: "Ercsi Tr.H." 'Ercsi trafóház
           end
KUT: begin
             K6: "Kút #K6" 'Kút UNIT
             'Kút #K6
           end

```

Figyeljük meg, hogy a TRAF0 csoport után nincs komment, ezért a fastruktúrában is csak a unit név jelenik meg: TRAF0. Érdemes tehát mindenhol egy beszédes kommentet hozzáírni.

2. Változók név szerinti és indexelt lekérése

Ezen a ponton az ismert *VariableOf()* függvény bővítményeiről van szó, ami a 3.3-as változattól vált elérhetővé:

```

xVariableCountOf:: long function (prj: optional shortstring)
xVariableOf:       pvar function (variablename: univ; extension: shortstring)
xSetVariable::     procedure (variablename: univ; extension: shortstring; value: univ)

```

Az x-szel kezdődő eljárás és függvények nem csak stringet fogadnak el változónév gyanánt, de egész számot is, ami index-nek minősül. Ezért a univ típus. Az index terjedelem:

```
0 .. xVariableCountOf(„projekt”)-1
```

További különbség, hogy az összes változóreferenciára hivatkozhatunk az *extension* argumentum révén és nem csak az Min, Max, L, H kiterjesztésekre, mint eddig. Például az összes változó újraengedélyezése az alábbi scripttel lehetséges:

```

for i=0 to xVariableCountOf(„Vispro1”)-1 do
  xSetVariable(i, xVariableOf(i,„IO”) and not(io_inhibited));
end

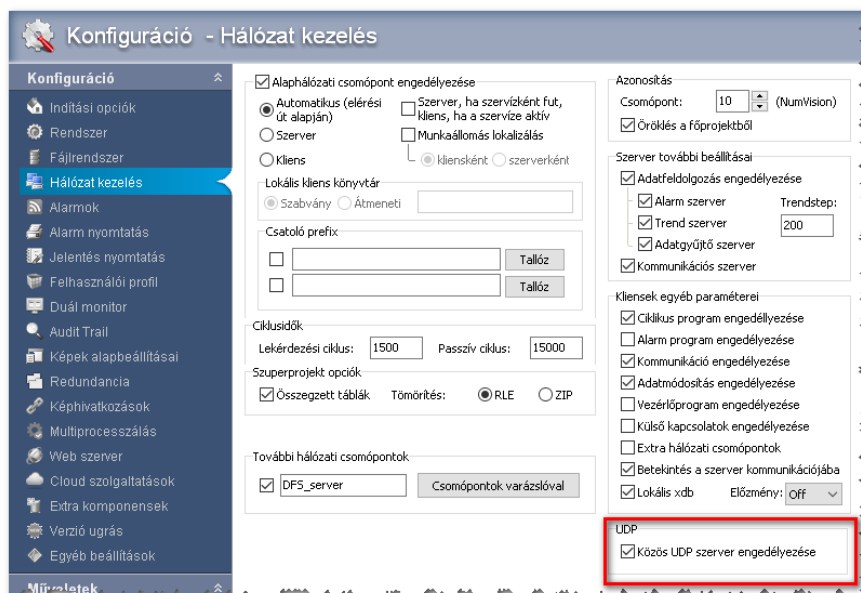
```

A fenti eljárás az Vispro1 nevű projekt összes változójának beolvasa az IO hivatkozását és törli belőlük a letiltást jelző bitet. Az új függvényekkel bármilyen a változók teljes halmazára érvényes vizsgálatot elvégezhetünk. Például letehetjük a változók értékeit és egyéb paramétereit egy adatbázis táblába...

X10 Verzió 3.2

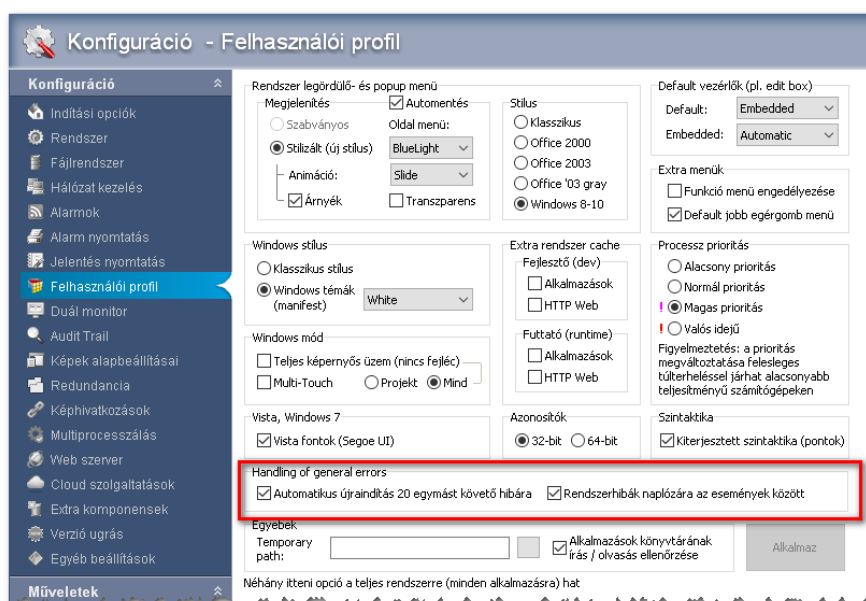
1. UDP server alkalmazáshoz kapcsolása

Ahogy a SAIA és az Omron kommunikáció leírásában is szerepel, ezek az eszközök közös UDP szerveren keresztül kommunikálnak. A mostani módosítás azt az igényt elégíti ki, hogy meghatározott szerver alkalmazáshoz köthessük ennek elindítását (és ne az elsőként indított szervíz / szerver alkalmazás ragadja azt magához). Beállítása a hálózat kezelésnél lehetséges:



2. Rendszer események naplózásának kikapcsolása

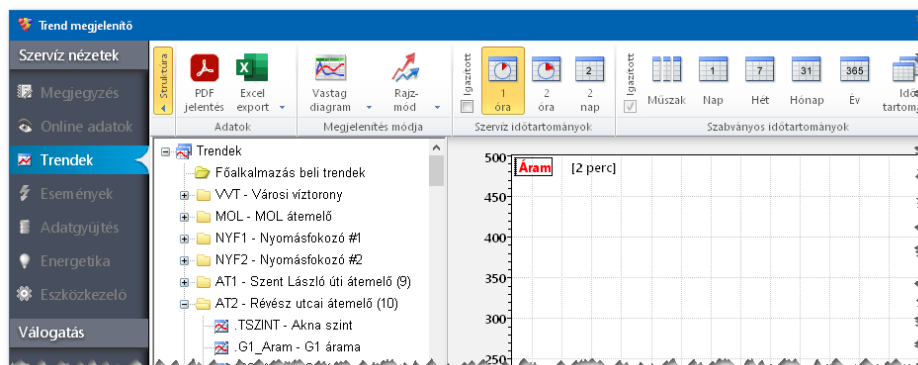
A rendszer hibajelzések naplózását tudjuk a következő két konfigurációs opcióval vezérelni és letiltani, hogy ne jelenjenek meg az alkalmazás esemény naplójában (ettől még a rendszer log fájlban az *error* alatt ott vannak), és ne is reagáljon rá a rendszer újraindítással:



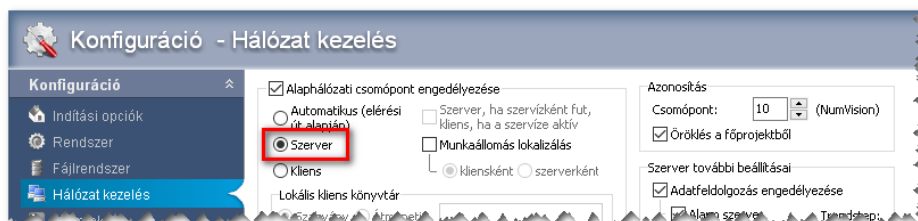
3. Kisebb hibajavítások

A program korábbi változatához képest történt sok apró javítás. Ezek a következők:

- 3.1. Javítottuk a kommunikációt a Saia PLC-vel.
- 3.2. Unitot tartalmazó alkalmazások változói strukturáltan jelennek meg mostantól a *TrendWindow*, *AlarmWindow* stb. képek bal oldalán, a fastruktúrában:



- 3.3. Forszírozott szerver hálózati alkalmazás mostantól akkor is szerverként indul el, ha egyébként belőle egy szerver példány már fut:



- 3.4. Nem hiba, de visszakapcsoltuk a szimbólum példányosításának lehetőségét (verzióugrás fül), mivel felhasználóink kérték.
- 3.5. MySQL-re állított adatgyűjtés nem inicializálta önmagát, ezt javítottuk.
- 3.6. MySQL trendek esetén automatikusan beállítjuk mostantól a Dátum-idő mező indexelését. Ez más, normális adatbázis kezelők esetén felesleges, mivel a dátum mezőtípus az adatbázis kezelőn belül eleve egy skaláris mennyiség: vagy hosszú lebegőpontos szám vagy egész. Ezeket általában felesleges indexelni. A MySQL ebben is kivétel.
- 3.7. DAC MySQL-nél nem működött a régi eseménynaplóban az interaktív szűrő – javítva.

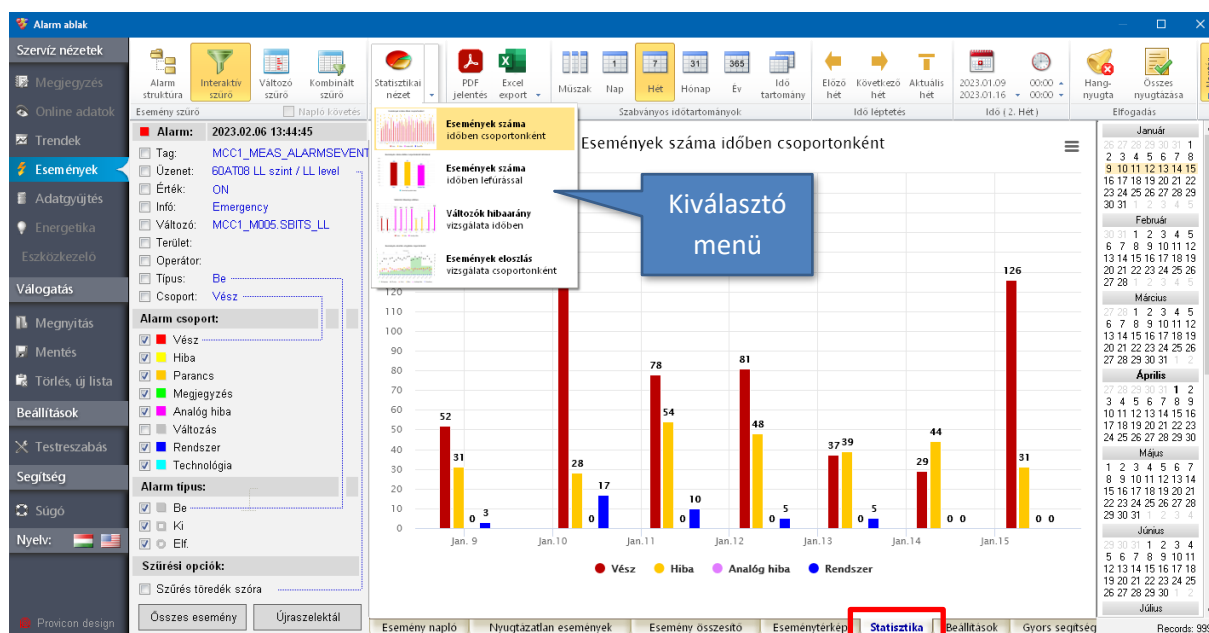
X10 Verzió 3.1-3.2

1. Alarm statisztika

A Highcharts-nak köszönhetően (ld. 3.0 változat leírását) az AlarmWindow-ba beépült egy esemény-statisztika és egy esemény térkép is.

1.1. Esemény statisztika

Az alábbi képen látható, hogy egy új fül (*Statisztika*) tartalmazza a szolgáltatást:



A lehetőségek az ábrán látható legördülő menüből választhatók ki:

- **Események száma időben csoportonként** – csoportonként összesíti az eseményeket és oszlopdiagramon megjeleníti az időtartománynak megfelelő felbontásban (napinál órás, hetinél napi, havinál és évesnél heti)
- **Események száma időben lefűrással** – annyiban tér el a fentitől, hogy az eseményeket a teljes időtartamra összegzi és az egyes oszlopokra való rákattintással adja vissza azok időbeli eloszlását (ez a lefűrás)
- **Változó hibaarány** – azt számítja ki, hogy az egyes változók mennyi ideig voltak hibában a bekövetkezés és a megszűnés közötti időket összeadva, majd az eredményt csökkenő sorrendbe állítva
- **Esemény eloszlás** – hisztogramot készít az események időbeli eloszlásáról; amennyiben az esemény bekövetkezése véletlenszerű, az egyes oszlopok magassága azonos, ha nem akkor vannak kiemelkedő valószínűségű periódusok (ezt másképp is feltérképezhetjük, erre szolgál a következő pontban ismertetett esemény térkép)

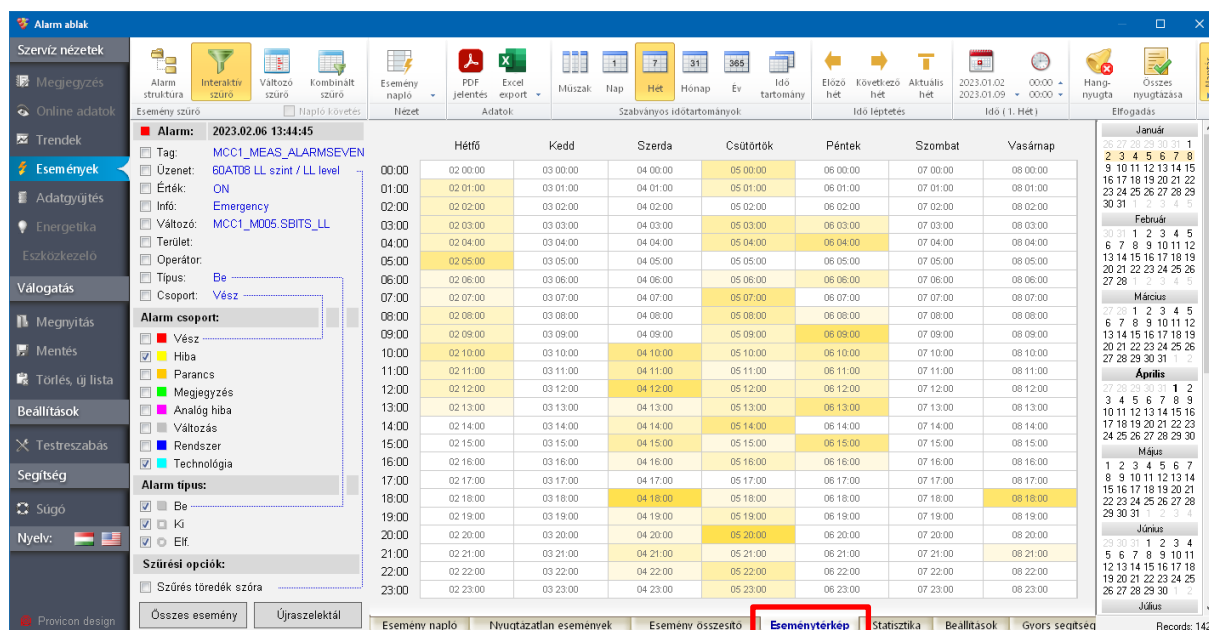
A fentiek közül talán a **változó hibaarány** a legérdekesebb, mivel ezt a számítást korábban soha nem végeztük el, noha eléggé fontos adatot közöl a berendezések állapotáról. Konkrét példákon tesztelve viszonylag gyakran találoztunk olyan berendezésekkel, amelyek az idő 50-70%-ban voltak hibásak(!), a rengeteg változó között azonban ez nem tűnt fel. Ezért olyan fontos a statisztika és a sorrendezés!

Az eseménystatisztika a következő képeket produkálja:



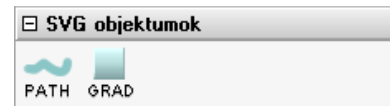
1.2. Esemény térkép

Az esemény térkép is egy új fülön található:



2. Műveletek SVG-képes objektumokkal

Az SVG-képesség azt jelenti, hogy a grafikus objektumból a VISION teljes értékű SVG képelemet tud konvertálni. Egyre több ilyen elemünk lesz, idővel minden lehetséges grafikus objektumra elkészül a konvertálás lehetősége. Azt hozzátéve, hogy persze soha nem lesz meg e képesség mindre, hiszen pl. egy *editbox*-nak, vagy egy *dropdown*-nak nincs SVG alternatívája. Egyelőre azonban csak a grafikonokra (chart, trend stb.) és a kifejezetten SVG céljára fejlesztett *Path* és *Gradient* (GRAD) objektumokra létezik a konvertálás, ahogy oldalt is látszik.

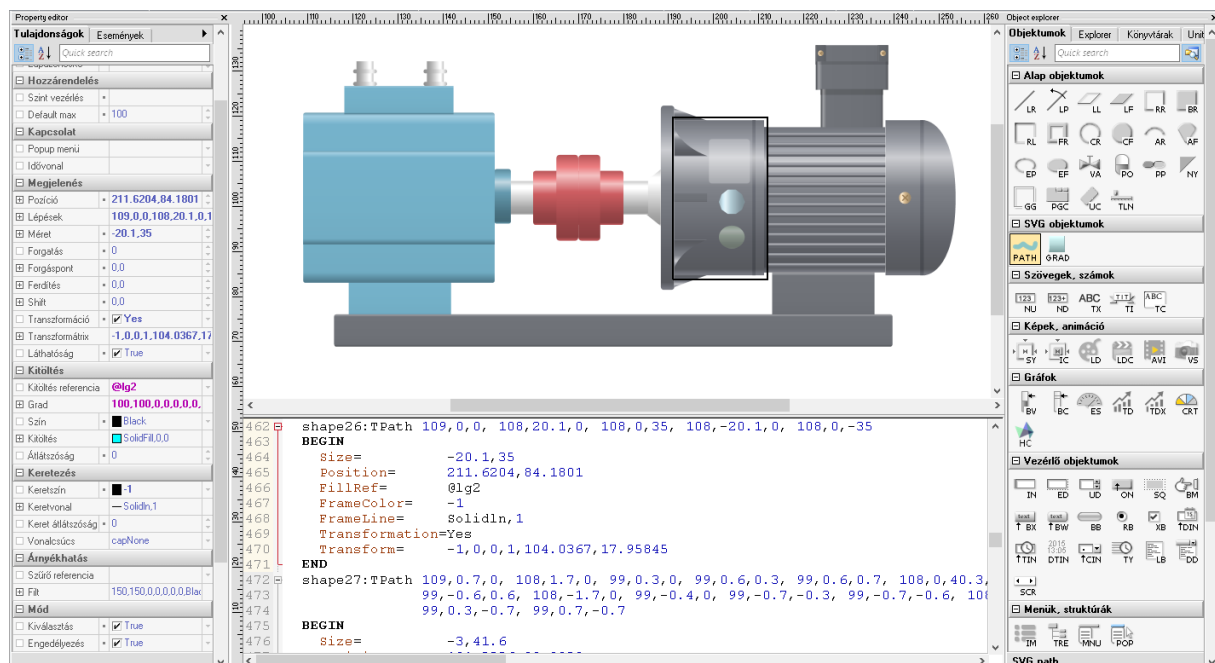


Mi az értelme az SVG képességnek? Miért foglalkozunk ezzel a kérdéssel egyáltalán?

A válasz egyszerű: a **látványos vektorgrafika** és a **web** miatt. Tallózóban ui. az SVG objektumokat lehet a leglátványosabban és a leggyorsabban megjeleníteni úgy, hogy nagyítás hatására ne pixelesedjen az ábra. Nem mindig igény, hogy az oldalunk tetszés szerint és torzításmentesen nagyítható legyen, de ha ez mégis felmerül, az SVG az egyetlen jó válasz jelenleg (A Canvas alapú HTML5 rajzok – ez lenne a másik lehetőség – sajnos olyan minőségvesztéssel járnak, különösen a szövegekre nézve, amit nem engedhetünk meg).

! Mielőtt tovább lépnénk, rögzítsük, hogy e témának nincs semmi köze ahhoz, hogy a VISION-ben meg tudunk-e jeleníteni SVG objektumokat vagy sem. Ez egy régi és továbbra is elérhető képessége a VISION-nek. Ebben a fejezetben a fordított irányt vizsgáljuk, amikor is VISION rajzokból készül SVG export.

Abban az egyszerű, ám eléggé gyakori esetben, ha csak *Path* és *Gradient* objektumokat használunk (ilyen mind a 4000 látványos vektorgrafikus szimbólum!), akkor egyszerű a helyzet: maga szimbólum SVG képes:



Az ilyen objektumokat, azok bármely részét, de az egészét is tetszőlegesen forgathatjuk, nagyíthatjuk, tükrözhetjük stb. Sőt mindenféle mátrix-transzformációknak is alávetethetjük.

A lényeg, hogy ezen objektumok felrajzolása a VISION-ben úgy történik, hogy először SVG objektummá konveráljuk a szimbólumot, majd abból az épp aktuális felbontásban egy PNG képecskét készítünk. A VISION pedig már csak a PNG-vel dolgozik. Ettől olyan gyors a program!

[...]	<DIR>	2023.04.01 14:03	----
Addictive_Pump_4271707129	svg	5,960	2023.04.01 14:03 -a-
Addictive_Pump_4271707129_86x80	png	5,614	2023.04.01 14:03 -a-
Air_lock_3970891808	svg	8,877	2023.04.01 14:03 -a-
Air_lock_3970891808_142x93	png	4,590	2023.04.01 14:03 -a-
Ammonia_separator_liquid_2001244467	svg	24,742	2023.04.01 14:03 -a-
Ammonia_separator_liquid_2001244467_126x201	png	8,260	2023.04.01 14:03 -a-
Asphalt_Filter_2701819493	svg	3,488	2023.04.01 14:03 -a-
Asphalt_Filter_2701819493_53x91	png	1,735	2023.04.01 14:03 -a-
Asphalt_Pump_4005252472_147x70	png	5,256	2023.04.01 14:03 -a-
Blower_room_2106516913	svg	11,092	2023.04.01 14:03 -a-
Blower_room_2106516913_88x70	png	2,804	2023.04.01 14:03 -a-
Boiler_780034181	svg	8,341	2023.04.01 14:03 -a-
Boiler_780034181_141x99	png	5,709	2023.04.01 14:03 -a-
Boiler_with_flames2_1149804876	svg	10,258	2023.04.01 14:03 -a-
Boiler_with_flames2_1149804876_113x122	png	6,050	2023.04.01 14:03 -a-
Draught_fan_4010958780	svg	6,803	2023.04.01 14:03 -a-
Draught_fan_4010958780_149x111	png	4,644	2023.04.01 14:03 -a-
Expansion_Tank_3690384906	svg	6,605	2023.04.01 14:03 -a-
Expansion_Tank_3690384906_116x205	png	6,079	2023.04.01 14:03 -a-

Ha valakit a részletek is érdekelnek, a VISION telepítési könyvtára alatt a Tmp\Png alkönyvtárban megtekintheti ezeket a képeket. Láthatóak az SVG képek és a belőlük adott méretben konvertált PNG képecskék is – egyeben.

A könyvtár tartalmát bármikor kiüríthetjük (sőt idővel érdemes irtani benne, mert nagyon sok fájl keletkezhet), a program legyártja, amit szükséges.

2.2. SVG dinamizálás szimbólumon belül

Mármost mi a helyzet a dinamizálással? Mi történik akkor, ha ezen objektumok bármely részét, annak színét pl. egy többállapotú változó attribútum táblájához kötjük? A válasz az, hogy az SVG ilyenkor más nevet kap (vegyük észre, hogy a névben szerepel egy hosszú szám – ez a content azonosító). És abból persze egy másik PNG képecske készül – végeredményben az összes színskombináció előáll. Ráadásul **a színezés képes megőrizni a gradiens színátmenetet!** A program baromi gyorsan képes váltogatni az egyes állapotok között. Történetesen még annál is gyorsabban, mintha natív VISION objektumokkal dolgoztunk volna!

Megjelenés	
☐ Pozíció	261.9005,118.594
☐ Lépések	109.0,14.8,99.0,0.9
☐ Méret	55.3,98.6
☐ Forgatás	0
☐ Forgáspont	0,0
☐ Fordítás	0,0
☐ Shift	0,0
☐ Transzformáció	☑ Yes
☐ Transzformátrix	1.0,0.1,2.9074,1
☐ Láthatóság	☑ True

Ennél sokkal bonyolultabb animációt is készíthetünk. Változtathatjuk az egyes részek láthatóságát, forgathatjuk az egyes elemeket változótól függően, állíthatjuk a forgáspontot, az eltolást (gondolj egy dugattyúra), sőt el is fordíthatjuk az objektumunkat. Ráadásul mátrix műveleteket is végezhetünk. Egyetlen dologgal nem érdemes bíbelődni: magával a [Path](#)-al (lépések), mivel az eléggé bonyolult.

A gyakorlati alkalmazásokban – az esetek 99%-ban – két animációs technika fordul elő az objektumon belül: szín és láthatóság.

Felmerül a kérdés, hogy rárajzolhatunk-e további objektumokat magára az SVG-képes objektumokra? (Például egy mért értéket akarunk kitenni NU-val, vagy egy bárgráfot, egy trendet stb.). A válasz az, hogy minden további nélkül! Ugyanúgy átadhatunk Xnumerics paramétert, rajzolhatunk aztán ezzel, amit csak akarunk. Működni fog.

Egyetlen fontos dolog van, és ezzel el is jutottunk e fejezet lényegéhez, **minden további objektumnak az SVG objektumok felett kell elhelyezkednie!** Más szóval, nem ágyazható be két SVG objektum közé. Mivel az SVG-készítés az első eltérő típusú objektummal vagy a fájl végén történik meg. Ennyi a lényeg, erre kell tehát egyedül ügyelni.

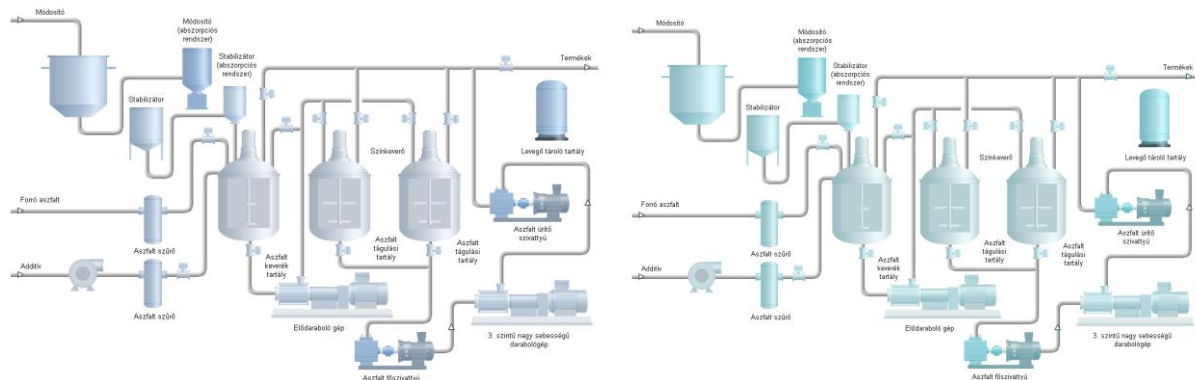
Az oka egyébként eléggé nyilvánvaló, mivel a program a szimbólum összes SVG-képes objektumából készít egyetlen SVG képet, nyilván ott lesz e transzformáció vége, ahol a szöveg, trend, bárgráf stb. kezdődik, vagy a szimbólum véget ér.

2.3. SVG dinamizálás szimbólumon kívül

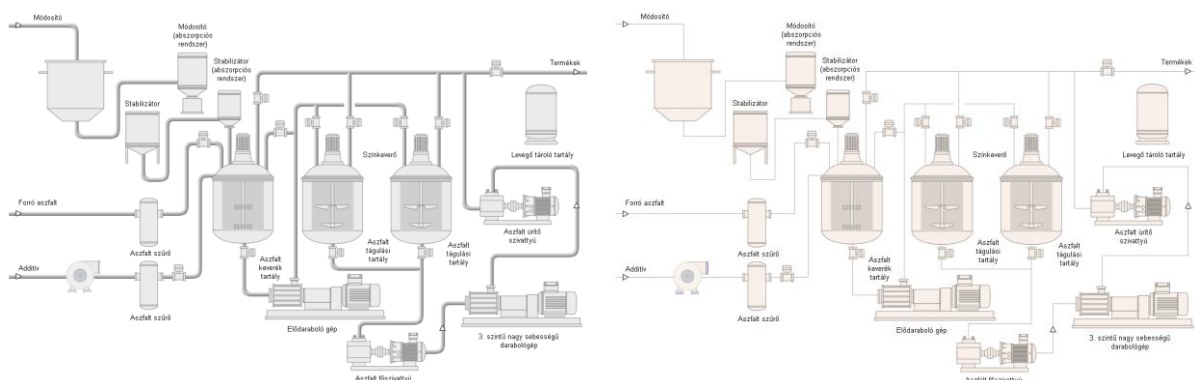
Az SVG képes objektumokra épülő szimbólumok úgymond kívülről, magában az SY vagy IC objektumban is átszínezhetők: *Sík kitöltés* és *Szín*. Ez látható jobb oldalt (SY megjelenítési tulajdonságai). Ezzel a módszerrel elérjük, hogy a szimbólumhoz akár hozzá sem kell nyúlni, hiszen a szín kívülről is átírható. Sőt még valami: a kitöltés módja, hogy ti. gradiens

Megjelenés	
☐ Pozíció	48.75+21.97.375-1
☐ Méret	6.875,14.25
☐ Láthatóság	☑ True
☐ Árnyék	☑ True
☐ Forgatás	0
☐ HotSpot X	0
☐ HotSpot Y	0
☐ Sík kitöltés	☐ Globals.flatFill
☐ Szín	☐ Globals.fillColor
☐ Kitöltés	solidfill,0,0
☐ Esemény frissítés	
☐ Cursor	crDefault
☐ Megi. vezérlése	

kitöltésű legyen-e. Mi több, el is forgathatjuk az SVG képes szimbólumot. A programban található példákkal érdemes kísérletezni. Egyetlen mozdulattal átszínezhető akár az összes objektum:

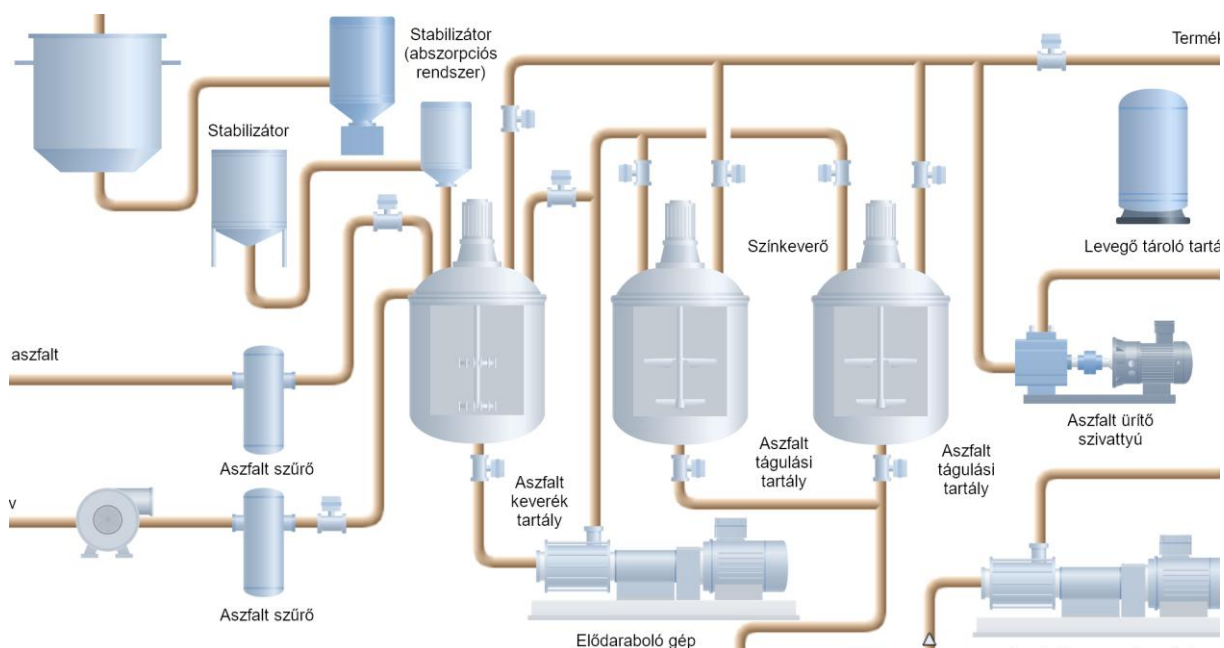


Sík kitöltést használva mérnöki vonal-ábrává alakíthatjuk a képet:



De még a csővezetékeket is átalakíthatjuk sima vonalakra, ami már a **Hi-Performance HMI** világába vezet el bennünket – és az ábráink még így is gyönyörűek! Többek között ezért érdemes hát SVG-képes objektumokkal dolgozni.

A fő ok persze – ha ez nem lenne még elég – a web: ránagyítva sem pixelesedik az ábra:



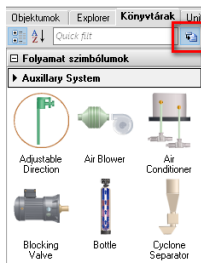
A web-es működésnek vannak más előnyei is: a méret. Jóval kisebb mennyiségű adatot kell mozgatni a hálózaton, ha egy SVG kép módosul, mintha a neki megfelelő PNG képet mozgattuk volna.



Ez a képecske pl. PNG képként 2035 byte méretű, és ugye a PNG eleve tömörített. Ennek SVG verziója pedig, tömörítve, 957 byte. Vagyis kevesebb, mint a fele. Ráadásul a képméret növekedésével ez az arány tovább csökken az SVG javára. Azért kell a tömörített arányt figyelembe venni, mert a web-re történő letöltés is így történik.

2.4. SVG lokalizálás

Az SVG képességű VISION szimbólumok a *Library*-ban találhatók. Mint a hagyományos szimbólumok esetén, itt is eldöntendő, hogy a szimbólumot bemásoljuk-e az alkalmazás könyvtárába. Erre akkor van szükség, ha módosítani szeretnénk, átrajzolni, vagy dinamizálni annak valamelyik belső elemét. Mert egyébként, ahogy a 2.3. pontban szerepel, az SVG képes szimbólumok úgymond kívülről is animálhatók, színezhethők – anélkül tehát, hogy azt lokalizálnánk. Feleslegesen pedig ne töltsük tele az alkalmazásunk könyvtárát szimbólumokkal, se hagyományosakkal, se SVG képességűekkel.



A lokalizálás az ábrán látható gomb benyomott állapotában történik

Bemásolás az
alkalmazás
könyvtárába

3. Kisebb hibajavítások

A program korábbi változatához képest történt pár javítás. Ezek a következők:

- 3.1. Javítottuk magát a manager alkalmazást és a kiegészítőket számos helyen. Különösen a kiegészítőkre tekintettel, ezeket mindenképp érdemes az alkalmazásokban frissíteni.
- 3.2. Dlong típusú analóg változók adatgyűjtése hibás volt, amennyiben az adat szorzófactora kisebb volt egynél. Ezt kijavítottuk.
- 3.3. Csak a demót érinti, de az AdvancedCourse alkalmazás mostantól megint XDB alarmmal és Audit Trail adatbázissal működik, ami a 64-bites Access-szel nem rendelkező gépeken fontos.
- 3.4. A 64-bites Access-szel nem rendelkező gépeken is működni fog mostantól az Access adatbázis kezelés a DBSRV processzen keresztül, ahogy a 64-bites rendszerben is volt. Ennek a megoldásnak a használata azonban továbbra is csak teszt jelleggel javasolt.

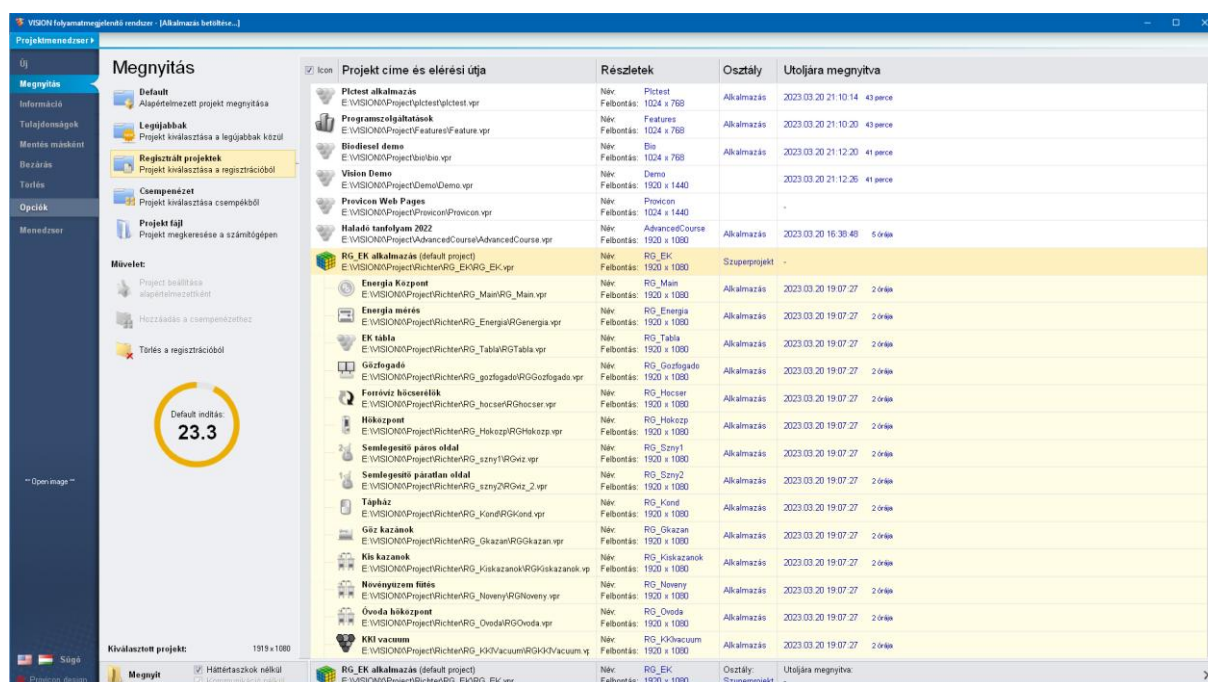
X10 Verzió 3.0

Új mérföldkönek számít a 3.0 verzió kibocsátása, mivel a rendszerben sok jelentős fejlesztés történt: Megjelent egy teljesen VISION alapú projekt menedzser és kibővült az *Xterior* szolgáltatás, vagyis a *ribbon* vezérlők VISION-ben implementált rendszere. Új szolgáltatás a *Highcharts*, a világ vezető chart rajzolójának teljeskörű VISION implementációja beépített objektumként. Bekerült a programba számos statisztikai funkció és azok felhasználása például az *AlarmWindow*-ban. Tovább bővültek az SVG-képes objektumokkal kapcsolatos műveletek, plusz számos további függvény és eljárás került még a rendszerbe. Nézzük meg ezeket részletesebben.

1. Projekt menedzser

Az új projekt menedzser a szokásos (például az *Office*-ban is megfigyelhető) trendeket követi, aminek lényege, hogy a projekttel kapcsolatos összes feladat – megnyitás, új projekt létrehozása, archiválás stb. – külön alkalmazási felületet kap. Ahogy az Excel-ben a *file*-ra, úgy kattinthatunk vissza a VISION-ban a *projekt*-re, amely tartalmazza az összes vele kapcsolatos műveletet.

Az új projekt menedzser látszik a következő ábrán. Figyeljük meg, hogy az összevont (szuper) projektek fastruktúrába szervezettek, azonosításukat a színezés is segíti:

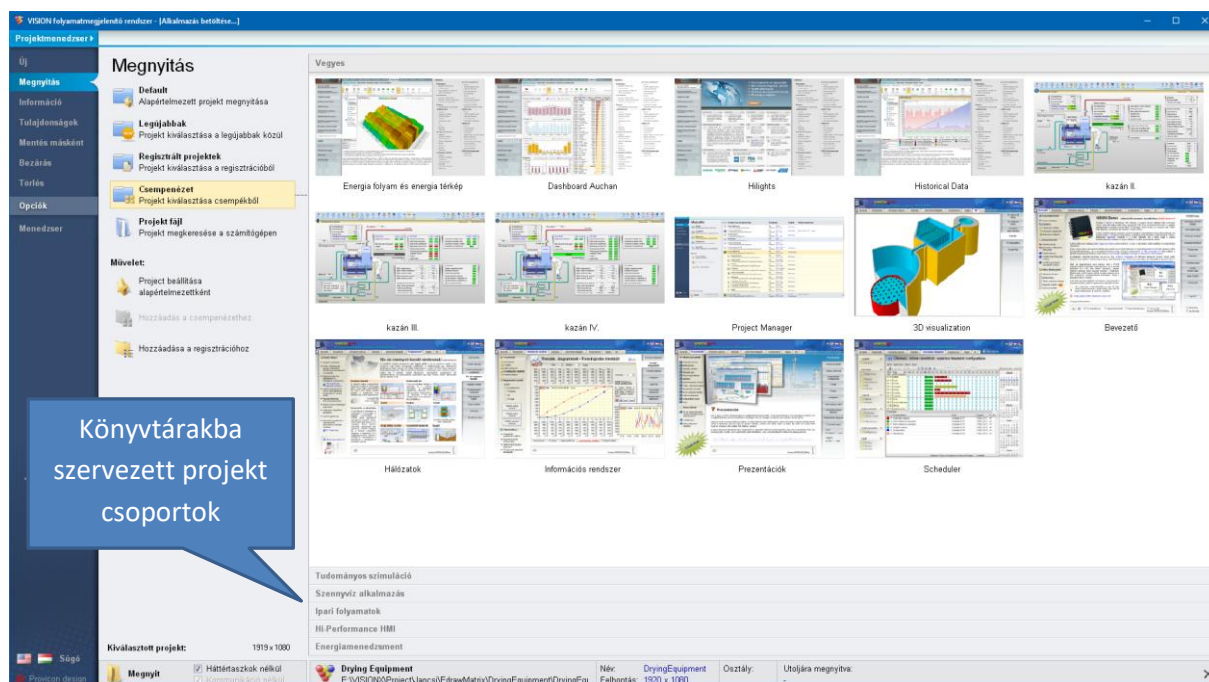


A projekteket az új rendszerben többféleképpen is listázhatjuk:

- Default projekt (csak egy sor)
- Legújabbak a megnyitás időrendi sorrendjében (*recent projekt list*)
- Regisztrált projektek listája (korábban csak ez létezett)
- Csempenézet (látványos grafikus tile-ok rendszere)
- Projekt fájl tallózó

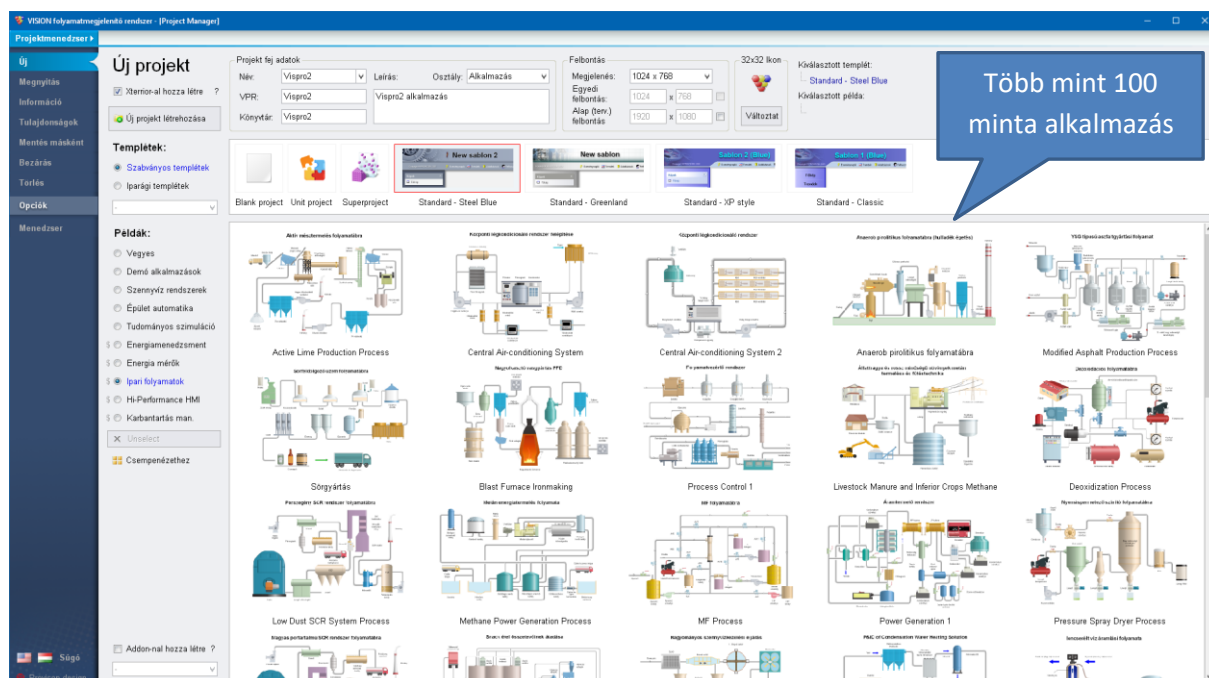
A listák is sokkal informatívabbak, mint a korábbiak, hiszen szerepel bennük a projekt 32x32-es ikonja, típusa, leírása és felbontása. Sőt még az is, hogy mikor lett utoljára megnyitva.

Nagyon látványos projektválasztó a csempenézet, amibe a projektjeinket egyetlen gombnyomással tehetjük be. Mindössze annyi szükséges hozzá, hogy a projekt azonosításra alkalmas képen (képein) megnyomjuk a Ctrl+K billentyűt. Ez legyárt a képről a *Preview* alkönyvtárban egy kis képecskét – a csempenézet pedig ezekből dolgozik:



A fenti képen minden egyes tile, csempe egy önálló projekt, amit így a képével azonosítunk. A csempenézet használatával megkönnyíthetjük a projekt kiválasztását.

A projekt menedzser azonban sokkal többet tud ennél: az egyes feladatokat a kép bal oldalán található oldalsó menüből választhatók ki. Ezek közül a legérdekesebb talán az új projekt előállítására szolgáló varázsló, amelyben több, mint 100 minta-alkalmazásból választhatunk és kombinálhatjuk azt több tucat templéttel. Mindezt megint kategorizálva, könyvtárakba szervezve:

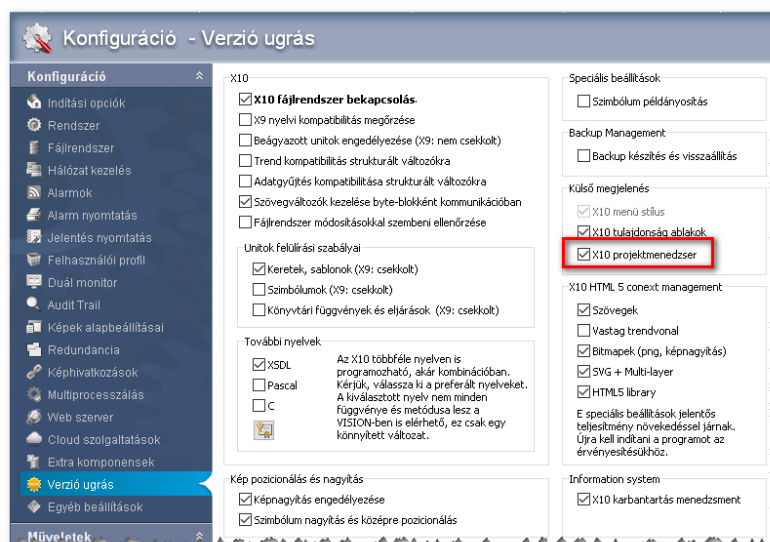


Az új projekt menedzser használatáról betanító videó is készült és bekerült az InfoVideo weboldalba.

https://www.provicon.net/Video/Vision/VISION_Prjman.html

Vision videos

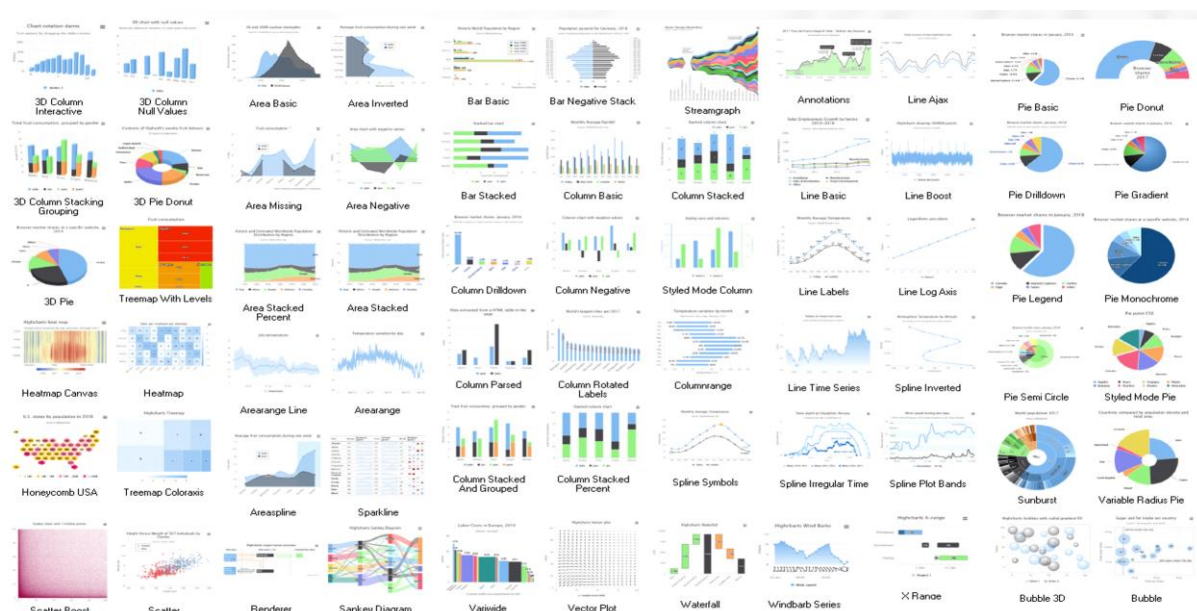
Itt lehet kikapcsolni az új projekt menedzsert és visszatérni az eredeti stílusra, ha valakit zavar:



2. Highcharts

A [Highcharts](#) a világ vezető chart (diagram) rajzoló komponense – licencelt termék. A megoldás érdekessége, hogy egyetlen objektumként lett beépítve a VISION-ba, ami így közvetlenül VISION változókkal vezérelhető. Az adatok származhatnak egyedi változókból, változó tömbökből, adatbázisokból és trendváltozókból is. Az grafikon elkészítése lényegesen egyszerűbb VISION objektumként, mint a chart JSON kódját előállítva, hisz ahány chart, annyi kód- és adatszerkezet van. A több ezer kombinációt a VISION kezeli helyettünk és számos automatizmust is tartalmaz.

Az alábbi ábra egy válogatás a leghasznosabb chartokból:



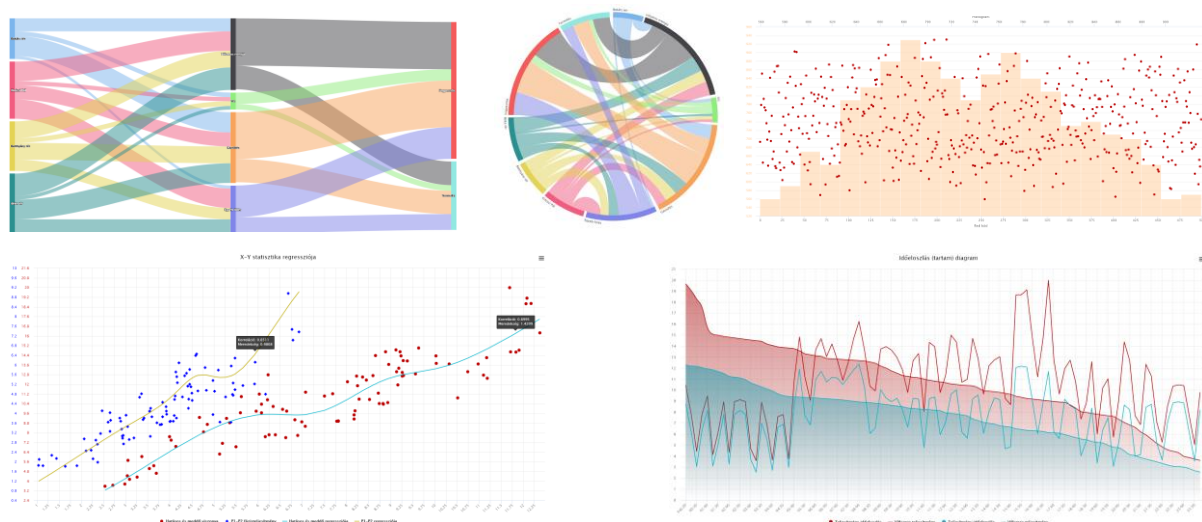
A Highcharts objektum (HC) használatához mindenképp először azt kell tudni, hogy az adatokat nem csak időben, de kategóriák szerint is meg tudja jeleníteni. Például alkatrész-típus, esemény, gép, vizsgálati tartomány stb. szerint. Maga az objektum nem bonyolultabb, mint az eredeti CRT komponens. Amíg azonban a CRT csak két adatsort volt képes megjeleníteni, a HC 8-at. Ezek a sorozatok.

A jobb oldali ábrán a HC objektum tulajdonságai láthatóak. Elsőként a chart típusát kell kiválasztanunk az ábrán itt legfelül látható listából (*hcAreaRange*). Közel 30 lehetőség közül választhatunk, amit az AdvancedCourse demo programban részletesen is meg lehet vizsgálni.

A chart az X és az Y tengely specifikálása után a sorozatok megadásával készül el. A sorozat chart-típusa eltérhet a főtípustól – ez a kombinált chartok esete, amikor például oszlopdiaagramot kombinálunk vonal-diagramokkal. A sorozatban megadhatjuk annak nevét, vonalvastagságát, vonalszínét és kitöltési színét, azt, hogy gradiens kitöltést használunk-e vagy egyenleteset, a minimum és a maximum értéket (ha nem adjuk meg, auto-skálázott a sorozat) és egy számítási módot, hogy ti. a sorozat adatait értékmegadással, számlálással, összesítéssel vagy átlagolással kell-e kiszámítani. Sőt egy feltétel is megadható, képlettel, hogy milyen elemeket vegyen figyelembe az adatbázisból a program egyáltalán. Maga a chart tehát tartalmaz bizonyos statisztikai műveleteket. Elsajátítása haladó tanfolyami anyag.

Érdemes még megemlíteni, hogy nem csak VISION változókra hivatkozhatunk a chartban, de az adatbázis mezőnevére is – úgy az X tengely megadásánál, mint a sorozatban. Ahogy a bevezetőben említettem, az X tengely nem csak dátum-idő típusú adatra hivatkozhat, de kategória-adatra is. Például a gyártmány típusára. Ilyenkor a chart összegyűjti az adott mezőnéven található adattípusokat és aszerint válogatja hozzá a sorozat elemeit. Ráadásul az így kiválasztott adatokat csökkenő sorba is állítja.

Egészen különleges chartok is készíthetők a VISION-nel. Lehetőség van lefűrészes (drilldown), amikor is egy összesített adat belső időeloszlását kapjuk meg rákattintással. A legkülönlegesebb talán a *Sankey* diagram, vagy az ún. függőség-kerék, de érdekesek az eloszlás diagramok is. Ám legfőképp hasznosak:



A chartok használatáról betanító videó is készült. Nem meglepően nem túl rövid sajnos:

https://www.provicon.net/Video/Vision/Highcharts_in_VISION.mp4

Vision videos

3. Adatbázis konvertálása statisztikai elemzéssel

Az adatbázis típusok közötti konvertálásra eddig is léteztek parancsok (fh / ConverData), de egy új eljárás használatával ez egyrészt sokkal gyorsabb lett, másrészt végezhetünk képlet szerint kiértékelést is a mezőelemek között. Maga az eljárás a *ConvertDBTable*:

```
ConvertDBTable (forrás_adatbázis, cél_adatbázis);
```

Legtöbbször XDB előállítására használjuk, mivel azzal rendkívül könnyen és gyorsan végezhetünk további műveleteket, de a cél_adatbázis lehet akár Excel fájl is.

Számításokat, statisztikai kiértékelést további paraméterek megadásával végezhetünk. A harmadik argumentumtól kezdve bárhány további mezőt készíthetünk a cél adatbázisban, így:

```
ConvertDBTable (forrás_adatbázis, cél_adatbázis),  
    "új_nező1=kifejezés",  
    "új_mező2=kifejezés",  
    ... ;
```

Például abban az esetben, ha egy további oszlopon a P1, P2 és P3 elnevezésű mezőink összegét szeretnénk kiszámítani, akkor ezt kell tenni (az új oszlop v. mező neve P123):

```
P123=P1+P2+P3
```

a teljes utasítás:

```
ConvertDBTable (source, dest, "P123=P1+P2+P3");
```

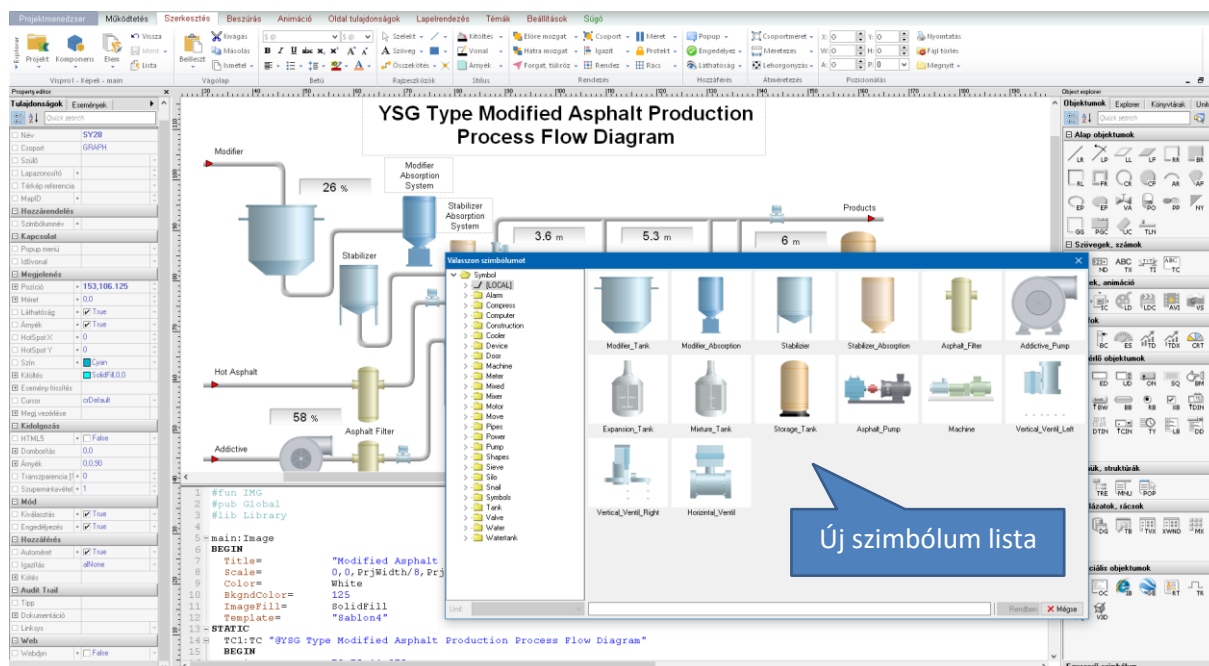
Úgy kell eljárni, ahogy Excel-ben csinálnánk. A kifejezésekben szerepeltethetünk zárójelet és függvényeket is. Közülük a leghasznosabb talán az **IF()** függvény, amivel feltételes kiértékelést végezhetünk. Például egy mező értékét csak akkor vesszük figyelembe, ha az vagy egy másik mező nem nulla, kisebb, mint egy megadott mező értéke stb. Azokat a függvényeket használhatjuk, amiket az SQL query feltételrészben is. Mindez azonban csak XDB cél adatbázis táblákra működik!

Kiváló példák találhatók a ConvertDBTable parancsra az AdvancedCourse demóban.

X10 Verzió 2.36

1. Szimbólum lista

A szimbólum lista egyrészt alpból nagyobb ablakban jelenik meg mostantól az egyre nagyobb felbontásokhoz igazodva, másrészt az összetett (control-, lista-, trend- stb.) objektumokat tartalmazó szimbólumokat is jól mutatja. A szimbólum választó jóval gyorsabb is lett. Ezt úgy érzük el, hogy a program egy PNG fájlt ment ki a szimbólumokból egy *Symbol* alkönyvtárba és a listában azt mutatja:



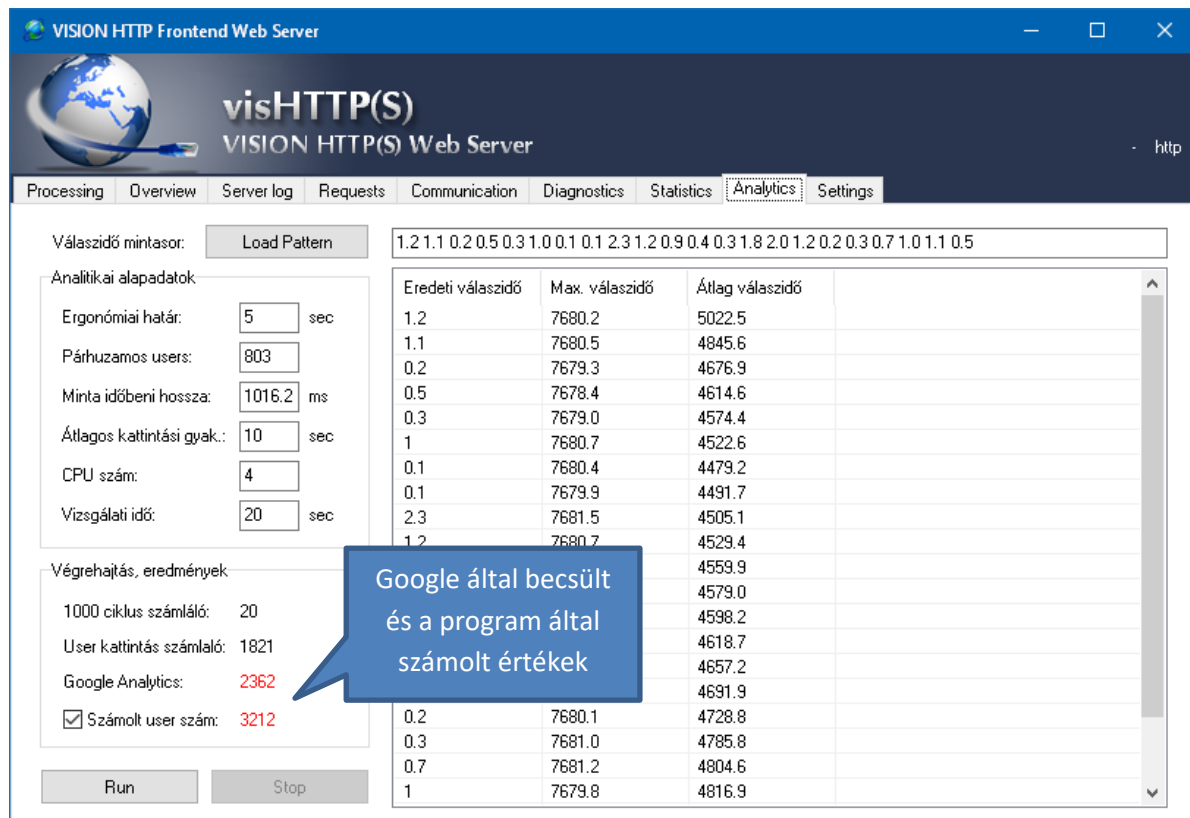
2. Web analitika

A visHTTPS frontend web szerver egy hasznos analitikai eszközzel bővült: valós adatokon alapuló statisztikai elemzéssel képes megállapítani, hogy az adott weboldalt hány felhasználó használhatja egyidejűleg – ergonómiai veszteség nélkül. Megfigyelhető, hogy amennyiben a frontend műveletek száma dominál (lila sorok) a backenddel szemben (kék sorok) – nem meglepő módon –, drasztikusan nő a weboldal teljesítménye, azaz annál többen használhatják párhuzamosan probléma mentesen.

A program alpból tartalmaz egy mintasort, amivel kipróbálhatjuk. A mintát a mérőkiolvasó miniSCADA program szolgáltatta, ott szinte kizárólag frontend kiszolgálás létezik. Nem meglepő, hogy ezzel a mintasorral a program 3000 párhuzamos felhasználó körüli eredményt produkál 4 CPU mag (azaz 1 virtuális gép) mellett. A hagyományos VISION alkalmazások azonban csak néhány tucatot.

A vizsgálat alapját a *Google Analytics* által kidolgozott algoritmus adja, azonban a program valós adatokkal operál. Így nem csak becslést kapunk, hanem pontos adatokat. Viszont nagyon hasonlókat a Google Analytics statisztikájához, amit több milliárd weboldal alapján dolgoztak ki. Mivel a visHTTPS programba épített eljárás hasonló eredményre jut a Google Analytics algoritmusához, a szimuláció validnak mondható.

Az eljárás lényege, hogy a diagnosztikai fülön indított felvétel során mért válaszidőket a program megjegyzi és akár több ezer párhuzamos felhasználó véletlen oldalhasználatával visszajátssza elemzi. A vizsgálat pár percet vesz igénybe, de a háttérben 1 ms-os ciklusidővel fut!



A becslés (Google) és a szimuláció alapján számított értékek (visHTTPS) az ábrán pirossal látszanak. A vizsgálat több (2-4) lépésben jut eredményre, amikor ui. az átlag válaszidő már nem haladja meg az ergnomómiai határértéket (5 sec). A program részletes ismertetése haladó tanfolyami anyag.

3. Bővítések, hibajavítások

A programban több kisebb funkció bővítés és hibajavítás is történt. Ezek a következők:

- 3.1. Tömbelemek bitjeinek is adható mostantól érték (`Tomb[12].2=true`), korábban ezek a hivatkozások csak az értékadás jobb oldalán szerepelhettek
- 3.2. Unit képek web-en történő teljes frissítése nem eredményez többé 404-es hibát
- 3.3. TrendWindow használatakor táblázatos nézetben, hálózati kliensen, az adat frissítésekor megváltozott a mutatott időtartam, ezt javítottuk
- 3.4. Ciklikus programok és könyvtári rutinok fordítása ritkán hibaüzenetet produkált, ezt is megszüntettük
- 3.5. Néhány hasznos új xdb funkció: egy rendkívül gyors szűrés két xdb fájl között és az alarm statisztika:

```
xdbCrossFilter::      procedure (sourcefield: shortstring; sourcedb: string; destfield: shortstring;...
xdbAlarmStatistics::  procedure (sourcedb: string; destdb: string; destdb2: optional string; options:...
```

Részletek a *Programming Reference Manual*-ban, ill. a rendszerleíró val.sdl fájlban.

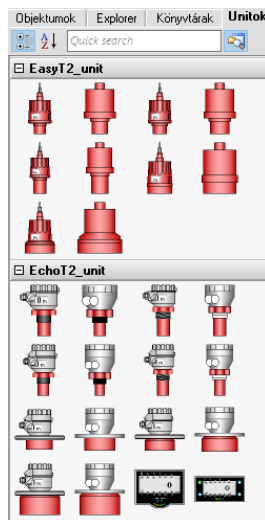
- 3.6. 40 karakternél hosszabb elnevezéseket nem tudott megjeleníteni a program a TrendWindow táblázatos nézetének oszlopneveiben. Mostantól ez a határérték 80 karakter. De azért törekedni kell arra, hogy ebbe beleférjen a megkülönböztetésre alkalmas változó leírás.

X10 Verzió 2.35

1. Hibajavítások

A programban több kisebb hibajavítás történt. Ezek a következők:

- 1.1. Szervíz üzemmódban megy a *RestartCom()* parancs (kommunikáció újraindítása)
- 1.2. SQL alapú adatgyűjtésnél (ACQ) az adatok visszaolvasása csak a megadott teljes méretig volt lehetséges. A javítással ez a korlát megszűnt.
- 1.3. *SimulateData* (adatgyűjtés feltöltése szimulációs adatokkal) parancs kiterjesztése SQL adatbázisokra
- 1.4. DAC és Cloud adatbázisoknál volt két apró hiba az UDLX fájl előállításakor (*ConnectionString* generálással nem volt probléma)
- 1.5. X10 üzemmódban a unit nem megfelelő méretben gyártotta le a szimbólumok ikonjait. A javítással ez meggyógyult:



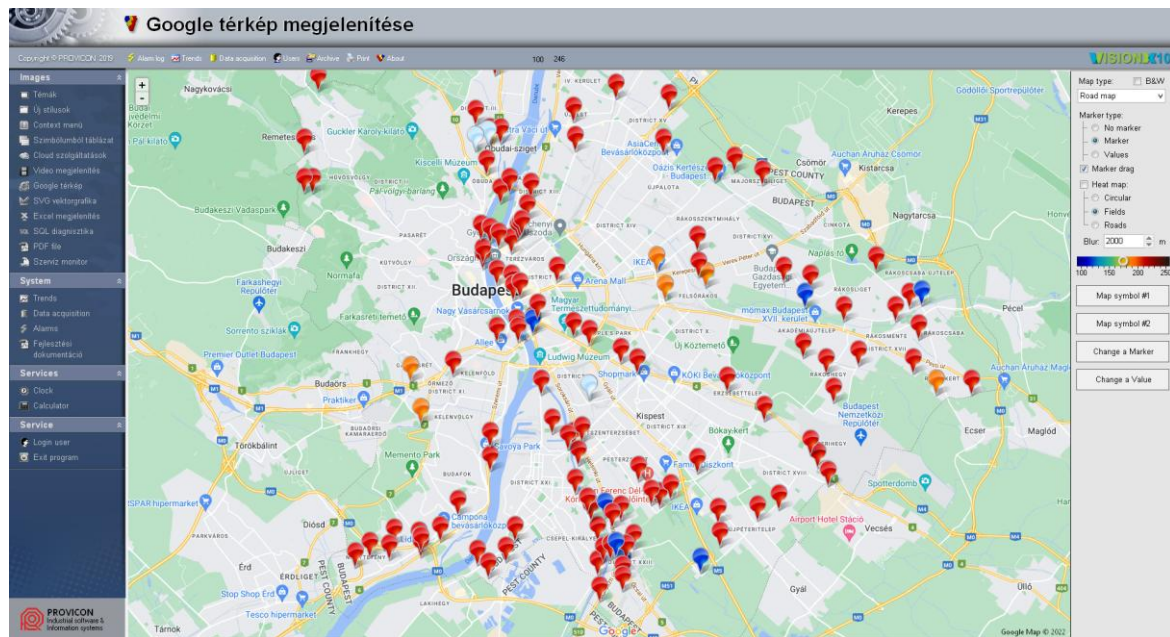
- 1.6. Strukturált változóhivatkozás javítása unit-on belül hívott ablakokban.
- 1.7. Strukturált változóhivatkozást tartalmazó tulajdonságok néha aposztrófok közé kerültek a grafikus szerkesztőben.

X10 Verzió 2.34

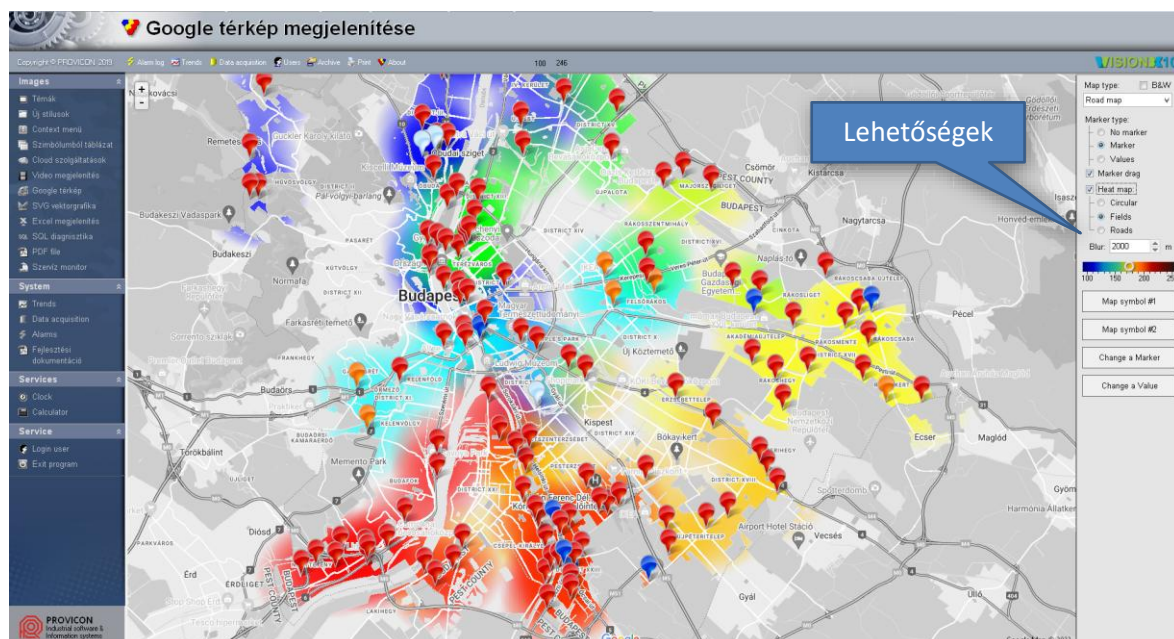
1. Google térkép használatának új lehetőségei

Eddig csak *Pinpoint* ikonokat (png) tudtunk elhelyezni a Google térképen az üzemegységek megjelenítésére, mostantól real-time VISION szimbólumokat is (**SY**, **IC**). De az új térképszolgáltatások még ennél is sokkal többet tudnak: valós időben változtathatjuk az ikonok színét és formáját, valamint a kiírt értékeket – túl az intelligens szimbólum megjelenítése miatti végtelen lehetőségeken. Ezen kívül megjeleníthetjük az adatainak hőtérképen is!

Az új szolgáltatások az *AdvanceCourse* demo Google térkép képén tanulmányozhatók:



A demó kép jobb oldalán láthatóak a megjelenítési opciók: térkép típus (*Road, Terrain, Hybrid...*), a marker típusa, a marker mozgatás (*drag*) és a hőtérkép:



A hőtérkép rajzolása is többféleképpen lehetséges: a pontok közötti területek színezésével (ld. fent), körkörös színezéssel és csak az utcák színezésével.

Innen továbbkattintva a *Map symbol #1*, vagy a *Map symbol #2* képre, a real-time szimbólum megjelenítésre láthatunk példát:



A térképek a jobb és bal egérgombokkal arrébb is mozgathatók, lökdöshetők. A plusz-mínusz ikonnal nagyítható és kicsinyíthető, ahogy a Google térképeken megszoktuk. Újdonság még a fekete-fehér térkép megjelenítés, ami azért lehet hasznos, mert jobban feltűnnek a színes ikonjaink és szimbólumaink.

Identification	
☐ Name	SY1
☐ Group	GRAPH
☐ Parent	
☐ PageID	
☐ Map reference	@XWND1
☐ MapID	117

A szimbólum hozzárendelés az SY és IC objektumok új *Map reference* és *MapID* mezőjével történik. Ez utóbbi arra szolgál, hogy a pontok adatbázisának FID mezője alapján meghatározza, hogy melyik pontról van szó a térképen. A program azután automatikusan odahúzza a szimbólumunkat, akárhová tettük is azt a képen szerkesztés közben. Vagy eltünteteti a vizibilitás automatikus változtatásával, ha már nem látható, mert a térképtartományon kívülre került.

A pontok adatbázisát (ami lehet XDB vagy CSV fájl is) hozzá kell rendelni a térképhez (*TableName*):

```
GOE1:GOE WindowWidth-46.125,WindowHeight-17.875
BEGIN
  TableName= "Facility2.csv"
  Fields= "Description", "Addr2", "GPS_lng", "GPS_lat", "SmallIcon", "", "Value"
```

És meg kell adni, hogy azon belül milyen oszlopnévvel ill. adatbázis mezővel hivatkozunk az adatra.

A Google térképek használatáról részletes oktató videó is készült – feltétlenül nézd meg:

<https://www.provicon.net/Video/VisionLessons/GoogleMaps.mp4>

Vision videos

2. Tömbváltozók átadása szimbólumnak és ablaknak

Az X10 paraméter típusai a tömbtípussal bővültek. Mostantól átadhatunk szimbólumnak és ablakoknak tetszőleges méretű tömbváltozókat is.

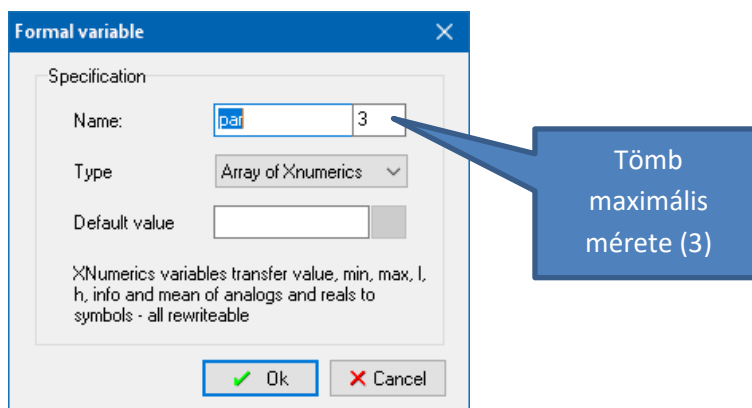
Először a tömbelemet kell deklarálnunk a megfelelő ablak vagy szimbólum formális változói között:

```
1 #fun IMG
2 #pub Global
3 #lib Library
4
5 ArrSym:Library Symbol (par[3]: Xnumerics)
6 BEGIN
```

A tömb maximális méretét így közöljük tehát a szimbólummal / képpel: szögletes zárójelek közé írva. Itt ügyelnünk kell arra, hogy a ténylegesen átadott változó tömb-mérete is legalább ekkora legyen. Ezután a *par* formális változóra már indexelve hivatkozhatunk a képen belül. Például:

```
19 NUI:NU par[0].frDip+njCenter+dmJoin+dmDark+dmSmall+exAlarm+exInhibit
20 BEGIN
21 Position= 22.5, 91.625
```

Természetesen a formális változó dialógusa is elvégzi számunkra a programírási munkát:



Megjelent tehát a legördülő menüben az *Array of Xnumerics*, *Array of Xdigits* stb. típus. Ha illet választunk, a név mellett a tömb mérete beviteli menü is aktív (ld. fent), egyébként nem látszik. Majd az Ok gombra kattintva a program beírja a scriptbe a paramétert a fent bemutatott szintaktikával.

Ahogy minden *Xnumerics*, *Xdigits* és *Xstrings*, a tömbváltozatok is írhatók, olvashatók és hivatkozhatunk az átadott tömbváltozó, mint objektum bármely további elemére (*min*, *max*, *factor*, *offset*, *dim*, *mean* stb.). Sőt a legtöbb elem írható is. Például:

```
Par[0].max=100;
```

A tömbparaméterek használatáról oktató videó is készült:

<https://www.provicon.net/Video/VisionLessons/ArrayParameters.mp4>

Vision videos

X10 Verzió 2.33

1. VISION adatbázis univerzum upgrade

A program [adatbázis kezelő](#) része újabb jelentős fejlesztésen ment át. Legfontosabbként azt lehet megállapítani, hogy az összes adatbázis kezelő illeszkedik a megfelelő SQL szerverek legújabb változataihoz. Részleteiben:

MS-SQL provider:

- SQL server 9 támogatás
- Azure AD autentikáció támogatás
- Azure “redirect” kapcsolatfelvételi mód támogat’asa “direct” módban
- Pár hibajavítás

MySQL:

- Apróbb hibajavítások

PostgreSQL:

- Megjelent a PostGIS támogatás
- TimeZoneOffset támogatás a TPgTimeStamp mezőtípusra
- Számos hibajavítás történt

SQLite:

- Direkt mód támogatása 3.39.2 adatbázis motor alatt
- Hibajavítások...

NexusDB:

- 4.60.01 változat támogatása

MS-Access:

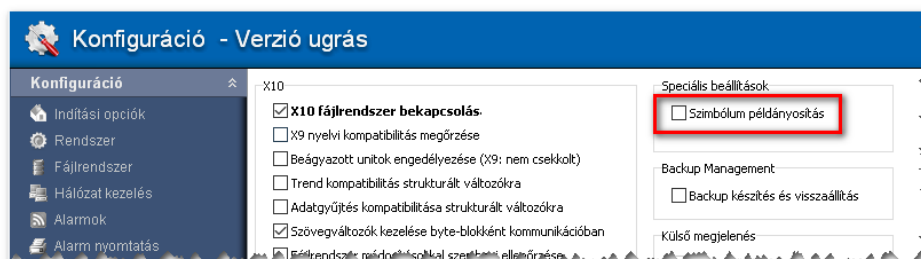
- Az UUID mező kezelésének a hibajavítása

InterBase:

- Timestep / time zone adattípusoknál történtek hibajavítások és számos további

2. Szimbólum példányosítás

Összetett szimbólumok esetén, amelyek önmagukban további szimbólumokat hívnak, azon belül újabbakat (3-szint) és nekik paramétert adnak át, előfordulhatott, hogy a szimbólum valamelyik példánya nem megfelelően frissült. Ezen segít egy új konfigurációs opció a verzió ugrás oldalon:



X10 Verzió 2.31-2.32

1. Új 1024 hosszú szövegváltozó típus

Néhány alkalmazásban kevésnek bizonyult az eddigi leghosszabb, 255 karakteres szövegtípus. Bevezettük ezért a *longstring*-et, ami már 1024 karaktert tud maximálisan tárolni:

The screenshot shows the 'Set string (message) variable' dialog box. The 'Type' dropdown menu is open, showing options: Longstring (1024-char), Shortstring (31-char), Widestring (255-char), and Longstring (1024-char). The 'Longstring (1024-char)' option is selected. The dialog includes sections for General data, Initialization, Data acquisition, Field access, and Publicity.

Elsősorban azokon a helyeken használható, ahol az ED (Editbox) objektummal hosszabb stringeket szeretnénk rögzíteni. Például a karbantartás menedzselő programban:

The screenshot shows the 'Defect' and 'Repair' forms in the maintenance management program. The 'Description' fields in both forms are highlighted with red rectangles, indicating the use of the Longstring type for longer text.

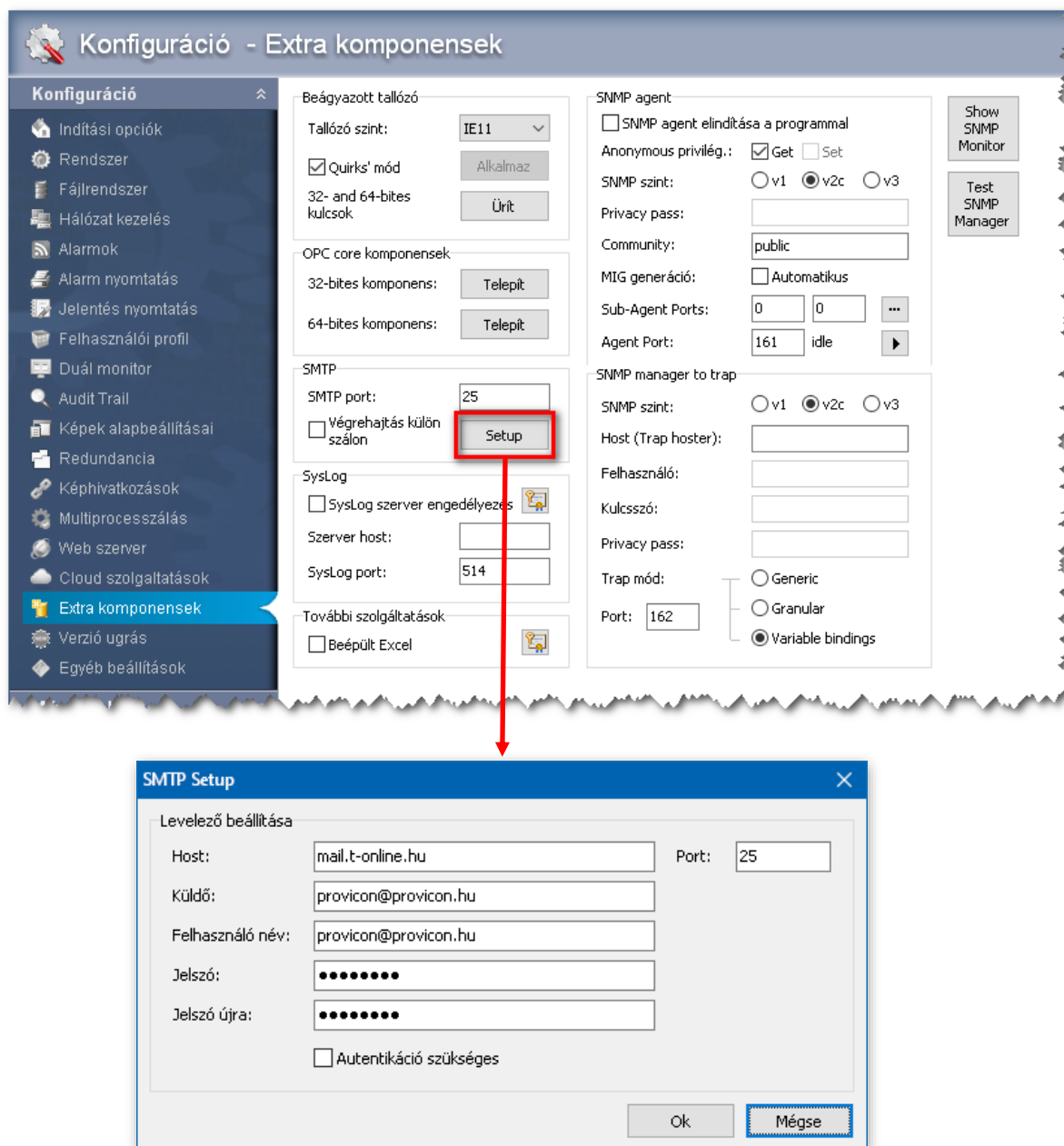
Lehet persze tárolni adatbázisban, onnan visszaolvasni stb. A *longstring* belül eleve UTF-8 formátumban kódolódik, ezért a tényleges karakterhossz – ékezetes karaktereket is feltételezve – kisebb lehet 1024-nél.

2. Kisebbs módosítások a kiegészítőben

Érdemes frissíteni a kiegészítőket, mivel számos apróbb módosítás történt (dual-monitoros környezetben való futtatás, alarm ikonok színezése, gyorsítás...).

3. Levelező globális beállítása

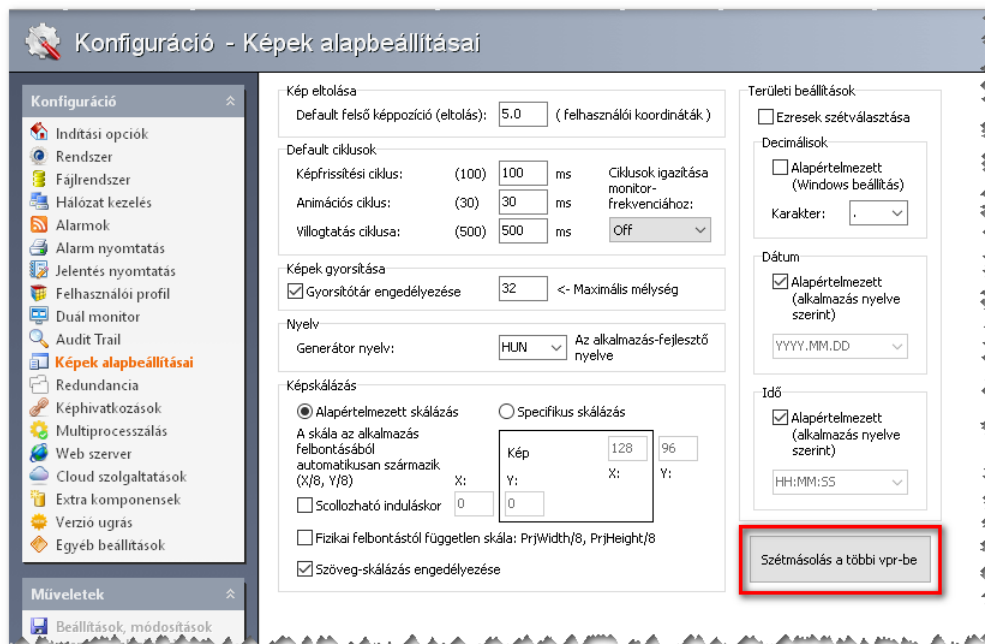
A 2.16-os változatban bevezetett *doMail()* parancs nem igényli a felhasználó név és jelszó megadását a levélküldés parancsban (ezért a forrásnak sem kell tartalmaznia felhasználó-specifikus információt), ellenben a vpr.ini fájlban kellett azt *kézzel* beállítani. A most bevezetett dialógus megkönnyíti ezt az eljárást.



X10 Verzió 2.30

1. VPR fájlok karbantartása

Egy kényelmi szolgáltatás lehetővé teszi, hogy egy adott projektfájl tartalmát szétmásoljuk a többi VPR fájl között – a konfigurációs rész nélkül. Erre akkor van szükség, ha ugyanannak a projektnek többféle nézete létezik, eltérő konfigurációval. Például van szerver, kliens és web kliens VPR. Ezek a leggyakoribb kombinációk. Ilyenkor fordul elő, hogy egy-egy új kép vagy szimbólum nem kerül át a többi VPR-be. S bár ezek karbantartása szövegszerkesztővel eddig sem volt túl komplikált, de a hibamentes megoldás a VISION képek alapbeállításai konfiguráció oldalán található, alul:



A gomb az aktuális projekt összes képét, struktúráit, adatbázisait, ablakait, sablonjait és szimbólumait másolja át az összes többi VPR fájlba, ami az alkalmazás könyvtárában található.

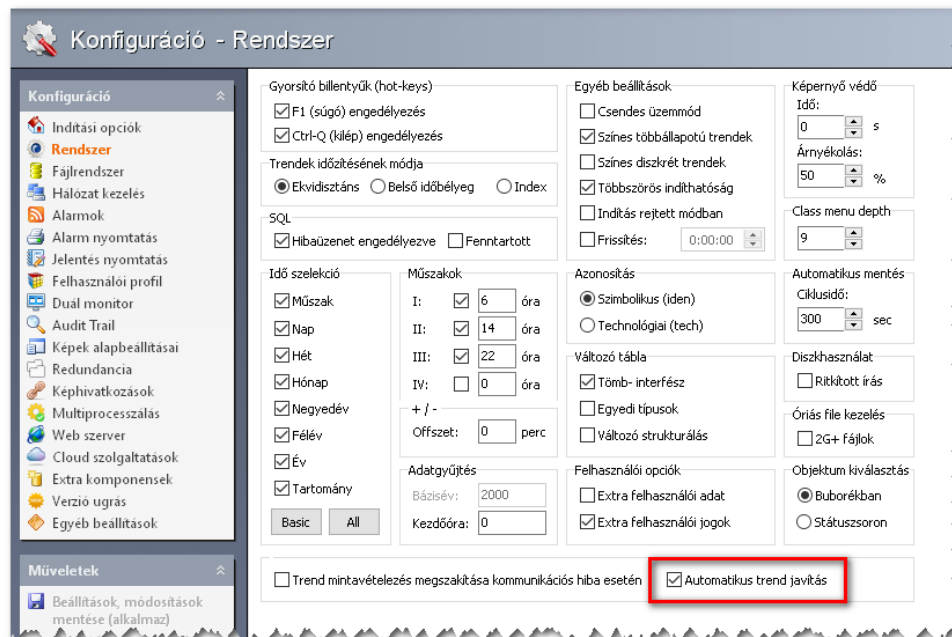
2. VISION adatbázis univerzum upgrade

A program [adatbázis kezelő](#) része újabb változáson ment át az előző verzió óta. Ezek főleg apróbb hibajavítások és további teljesítmény növelő fejlesztések voltak. Mindenképp érdemes az új verzióra upgrade-elni. A változtatás a teljesség igénye nélkül a következő adatbázisokat érintette: MS-SQL, Oracle, MySQL, PostgreSQL, Interbase, SQLite, MongoDB.

3. Trend és adatgyűjtés adatbázis hibák automatikus javítása

Ismeretes, hogy hibás rendszerleállítás, gép ki-bekapcsolás hatására, kis valószínűséggel bekövetkezhetnek fájlhibák a trend (.TRN) és az adatgyűjtés (.DAT) fájlokban, amit azután csak az archívumból való visszaállítással tudtuk javítani. Sajnos a művelet adatvesztéssel járt.

Ezen segít egy automatikus hibajavító funkció, amit a rendszer konfiguráció oldalán kapcsolhatunk be:

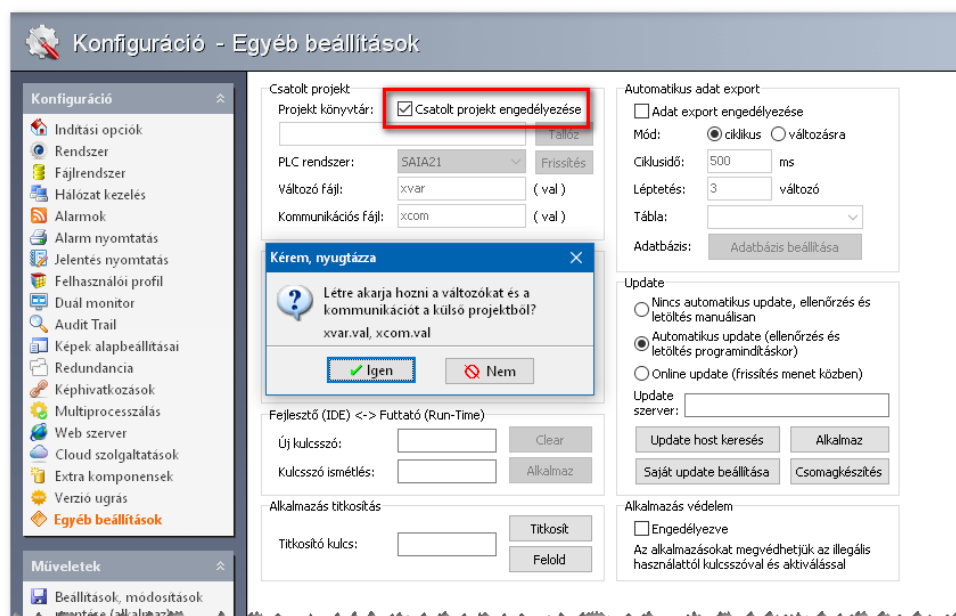


Természetesen ez sem véd minden hibától (a fájl lenullázódásától pl. nem véd), de a leggyakoribbakat – minimális adatvesztéssel – kezelni tudja.

Emlékeztetnék, hogy SQL trend és adatgyűjtés használata esetén ezek a hibák nem tapasztalhatók.

4. Projekt generálás SAIA PLC-ből

Hibajavítás és némi okosítás után ismét használhatjuk ezt a hasznos funkciót:



5. Felhasználó által konfigurálható Excel jelentések

Egy érdekes új funkció segítségével lehetőség van arra, hogy a VISION-ből – alkalmazásoi program fejlesztés nélkül! – olyan XLSX riportokat készíthessünk, amit a felhasználó tervezett meg. Ezáltal a felhasználó annyi és olyan riportot készít magának az adatokból, amelyet és amennyit csak akar.

A megoldás alapgondolata egyszerű, kivitelezése azonban komoly programozói munkát igényelt: az alkalmazás egységes szerkezetű excel outputot, outputokat készít, amire egy másik excel fájl külső cellahivatkozással hivatkozik, ez a sablon. A sablon lesz a jelentés maga, amit a VISION csak megnyit, a hivatkozásokat az Excel fejt ki. A sablon úgy formázható, alakítható, benne cellaműveletekkel, grafikonokkal stb., ahogy csak szeretnénk. És annyi sablon készíthető különböző célra, amennyit óhajtunk – további VISION programozás nélkül. Ez az alapgondolat.

Azonban jelentés készítésre ez az elv önmagában hibás, mivel **a sablon mindig ugyanarra hivatkozik**, ergo nem lesz annak különböző tartalma, hiába nevezzük át kimentés előtt. Egy új jelentés készítése megváltoztatná a régit is.

Ezt a dilemmát oldja meg az

```
OpenExcelSaveAs:: procedure (source: string; destination: string)
```

eljárás, ami belül a külső hivatkozásokat kifejt és olyan excel fájlt ment ki, ami már semmilyen külső hivatkozást nem tartalmaz, ergo nem is függ a tartalma önmagán kívül semmi mástól.

Tehát a VISION mindig ugyanabba az excel fájlba exportál mindig. Az *OpenExcelSaveAs()* függvény pedig kiértékeli a forrás sablon (*source*) hivatkozásait, behelyettesítve a képletekbe a hivatkozott konkrét adatokat az összes cellára néve és úgy menti ki tetszőleges néven (*destination*). Ezért volt olyan komplikált a megoldás, hisz implementálni kellett VISION-ben az excel képletkiértékelést és benne az adatok azonosítását.

Ehhez a feladatkörhöz tartozik egy másik hívás:

```
GetExcelReferences:: procedure (source: string; destination: string)
```

ami egy adott excel fájl külső hivatkozásait tárja fel és teszi ki egy XDB fájlba – ha bármilyen célból azt VISION-ben tudni szeretnénk. Például azért, hogy milyen kimenetek előállítását kell elvégeznünk:

Subdir|References

Ref|Földgáz elszámolószámla

Ref|Földgáz kapacitás elszámolószámla

Ezt beolvasva a *References* mezőre, megkaphatjuk a sablon (*source*) külső hivatkozásait.

Ha például egy excel sablon a következő hivatkozást tartalmazza:

=c:\Ref\[adat1.xlsx]Struktura'!\$W\$4*'c:\Ref\[adat2]Struktura'!\$AS\$4

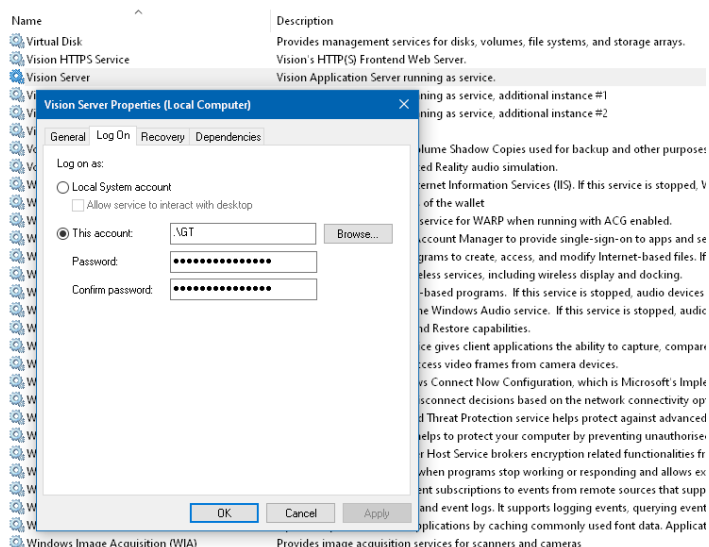
Akkor az adat1.xlsx és az adat2.xlsx fájl előállítása után meghívva az *OpenExcelSaveAs()* függvényt az alábbi cellabejegyzést kapjuk:

=21*0.3

Feltéve, hogy az adat1 fájl W4-es cellájában **21** volt, az adat2 fájl AS4 cellájában pedig **0.3**.

6. Szervízek beléptetése adott felhasználóra

A VISION szervíz üzemmódban történő futtatásához szükség lehet a szervíz adott felhasználóra (általában technikai felhasználóra) való beléptetésére. Ezt azonban csak kézzel tudjuk elkövetni a Windows szervízek listájában:

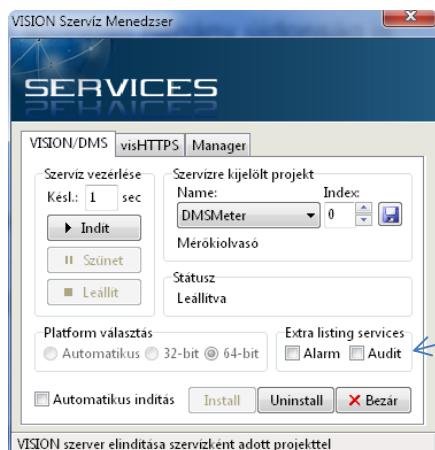


Környezettől függően ezzel elérhetjük, hogy a VISION biztosan hozzáférjen az összes szükséges szolgáltatáshoz – például az adatbázisokhoz (ld. Alarm adatbázis korábban vázolt problémáját).

Azokban az esetekben, amikor a második, harmadik stb. VISION szervíz példány már nem volt képes elérni az alarm adatbázisát, ez a beállítás mindig segített. De szükséges ahhoz is, hogy a felhasználófüggő beállításokat, például a dátum-idő formátumát a szervíz számára így előírjuk.

Egyetlen hátrány van csupán: mindenképp rendszergazda kell hozzá, hogy a műveletet elvégezze, mivel szükség van a megfelelő jelszó ismeretére.

Kollégák jelezték, hogy a szervízekről szóló korábbi leírás egy része, ami a *SetupServices* szolgáltatásról szól, nem teljesen egyértelmű, ezért javítottunk a szövegen és itt megismételjük:



A beállítás alapvetően nem változott, továbbra is az Index léptetésével lehet hozzárendelni az egyes szervízpéldányokhoz a projektet, maximum 10-et. Nincs ugyanakkor automatikus mentés – ezt a felhasználók kérték! – csak a mentés gomb megnyomásával lehet lementeni a beállításokat, külön az egyes indexekhez. Van azonban néhány újdonság is.

Extra listing services:

- Alarm
- Audit

A *VSM.ini* fájlba a rendszer ezeket a beállításokat *AlarmList* és *AuditTrail* néven menti el. Szerepe pedig az, hogy amennyiben nem kérjük, ezeket a szolgáltatásokat a szervíz el sem indítja.

Ez az opció az alarm szervízt kikapcsolja, lévén az alarm listázónak része. Nélküle nem fut az alarm taszk sem. Ami a listázó funkciót illeti, szervíz szinten megjeleníteni sem az alarm listát, sem az Audit Trail-t nem lehet, ezért emiatt önmagában nem szükséges a szolgáltatás, legalábbis a kliens szervízekben, de a szerver szervízben szükséges, mert különben nem keletkeznek alarmok. A kliens szintű alarm funkciók (nyugtázás, bejelentkeztetés stb.) azonban működnek nélküle is.

X10 Verzió 2.29

1. VISION adatbázis univerzum upgrade

A program [adatbázis kezelő](#) része jelentős fejlesztésen ment át és néhány további adatbázis kezelővel is kibővült. Legfontosabbként azt lehet megállapítani, hogy az összes adatbázis kezelő illeszkedik a megfelelő SQL szerverek legújabb változataihoz. Részleteiben:

MS-SQL provider:

- SQL server 2019 támogatás
- LOB (large binary objects) írás-olvasás sebességének jelentős gyorsítása

Oracle provider:

- Oracle 20c verzió támogatása
- PrefetchRows opció megvalósítása
- Adat olvasás és LOB írás-olvasás sebességének gyorsítása

MySQL provider:

- OpenSSL 1.1 támogatás!
- Tárolt eljárások gyorsítása

PostgreSQL provider:

- PostgreSQL 14 támogatás

SQLite provider:

- *LockingMode*, *Synchronous* és *JournalMode* implementálása
- Performancia jelentős javítása

MongoDB provider:

- *SQLEngine* opció implementálása

InterBase provider:

- *CutOffParams* és *WireCompression* opció implementálása
- Firebird 4 támogatás

NexusDB provider:

- NexusDB 4.50.26 verzió támogatás

Mindegyik adatbázis providerben történtek kisebb-nagyobb hibajavítások. Az új adatbázis providerek a következők:

BigQuery A Google Cloud platform egyik új eleme:
<https://www.youtube.com/watch?v=CFw4peH2UwU>

HubSpot Egy újabb CRM adatbázis platform a meglévők mellé
<https://knowledge.hubspot.com/get-started/manage-your-crm-database>

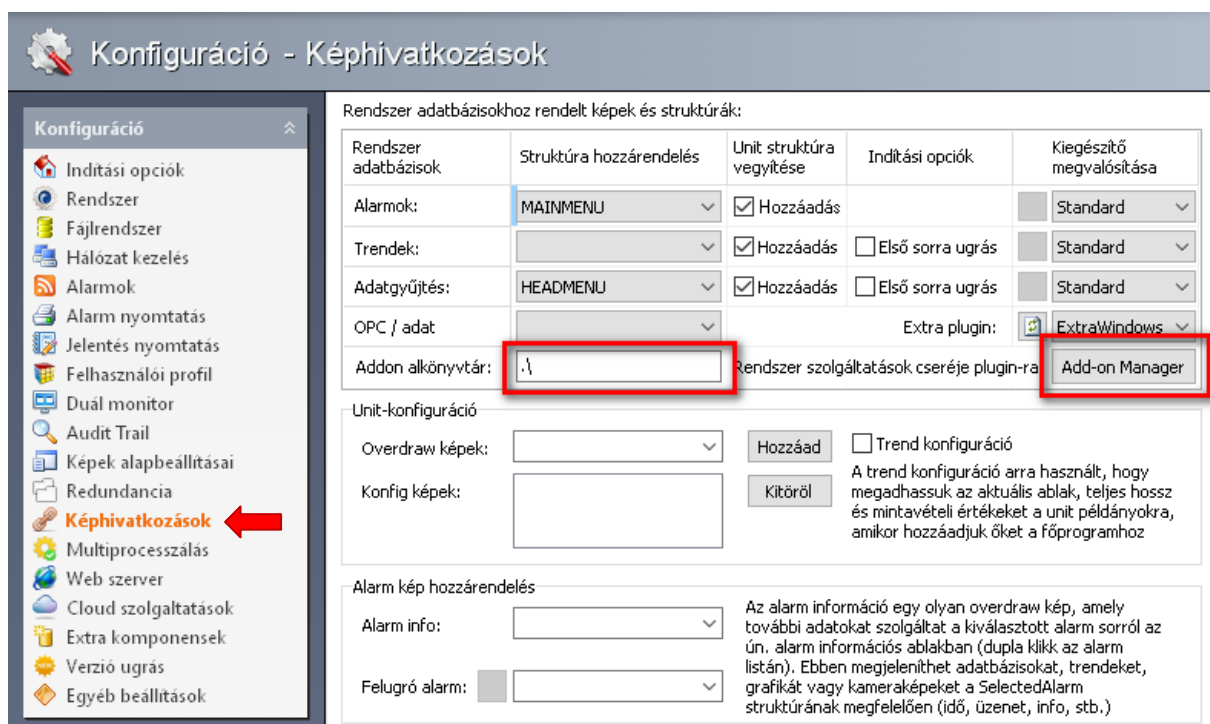
X10 Verzió 2.28

1. Add-on menedzser

A program az *Add-on* funkcionalitást illetően (*TrendWindow*, *AlarmWindow* stb.) egy rendkívül hasznos menedzser szolgáltatással bővült, amivel sokkal könnyebben konfigurálhatjuk mostantól a rendszert. Emlékeztetőül, a kiegészítők VISION alkalmazás fejlesztéssel (képrajzolással) megvalósított rendszerfunkciók: trendek listázása, esemény kezelés, adatgyűjtés stb. E megoldás egyetlen más SCADA rendszerben sem található meg (mivel ahhoz sokkal komolyabb képességekre lenne szükség). A kiegészítők bevezetésével minden korábbi korlát felszabadult, hisz a források birtokában bárki úgy módosítja, formálja, színezi az egyes ablakokat, változtatja a betűméretet stb., ahogy csak akarja.

Figyelem: az összes kiegészítő megváltozott, azokat frissíteni kell!

Az új funkció a konfigurációs lapon a képhivatkozások alatt található:

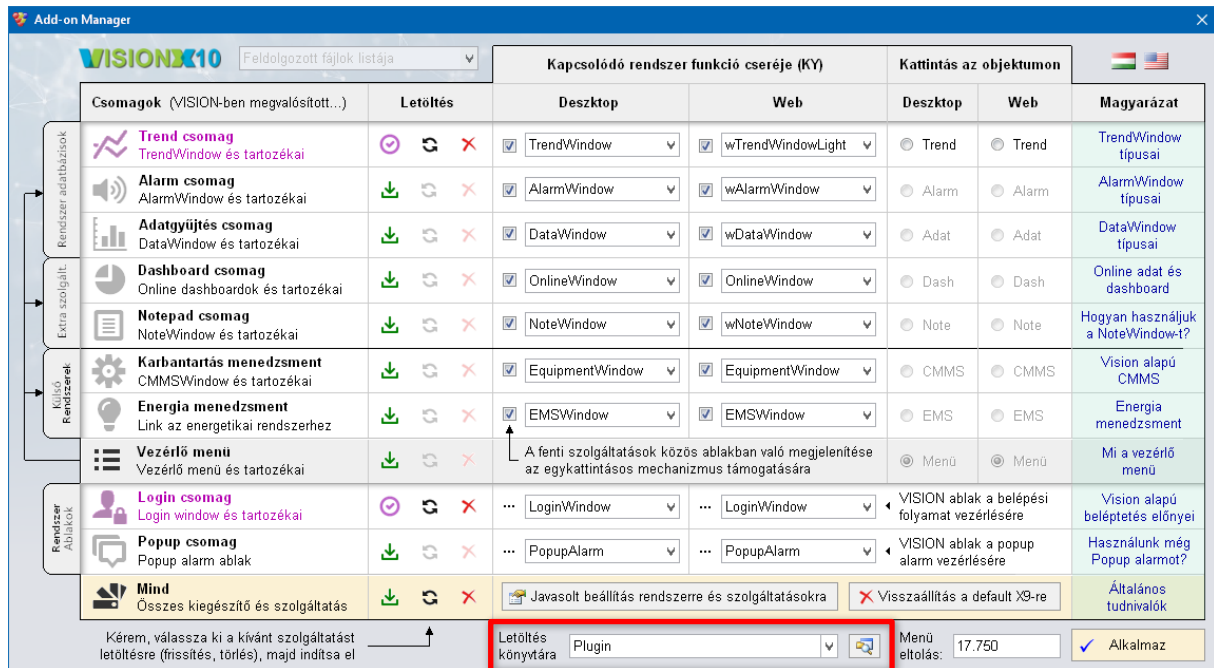


Egyrészt beállítható a kiegészítők **alkönyvtára**, tehát nem kell többé összekeverni a kiegészítőket az alkalmazással. Mivel rengeteg fájlról van szó, ez egyáltalán nem mellékes! Ráadásul az Addon könyvtár elhelyezkedhet egy szinttel feljebb is (..\Addon). Összevont alkalmazások tehát ugyanarra a könyvtárra és fájlrendszerre hivatkozhatnak – nem kell a kiegészítőket mindenhol bemásolni. Így nem csak tárhelyet spórolunk, de a frissítés is sokkal egyszerűbbé és áttekinthetőbbé vált.

Az *Add-on Manager* gomb fölötti kontrollokkal továbbra is megoldható az alarm-, trend- és adatgyűjtés letöltése, sőt frissíthetők is onnan – vagyis a rendszer kompatibilis a korábbi változattal.

Az *Add-on Manager* azonban egyetlen képernyőbe sűríti az összes funkciót. Ráadásul nem csak az egyes kiegészítőket választhatjuk itt ki, de frissíthetjük, törölhetjük is őket, továbbá beállíthatjuk, hogy deszktopon és weben mely kiegészítő jelenjen meg és mikor.

Maga az *Add-on Manager* egyetlen konfigurációs képből áll (és persze VISION képrajzolással készült):



Az első ránézésre bonyolult kép használata rendkívül egyszerű: az egyes sorokhoz tartozó  letöltés gombokat kell megnyomni, amikor eldöntjük, hogy mely kiegészítőket szeretnénk használni az alkalmazásunkban. Először csak kijelöljük, utána megerősítjük és letöltjük. Előtte állítsuk be a letöltés könyvtárát (alul)! Mivel a felhasználók rendszerint az összes kiegészítőt letöltik, a legutolsó (sárga) sorban ezt is felkínálja a program (**Mind** = összes kijelölése, majd letöltése):



Ennek hatására válik láthatóvá a sárga **Letöltés** gomb  a képernyő alján. Először tehát kijelöljük csak az egyes elemeket, majd egyetlen gombnyomással valóban le is töltjük őket. Az egész művelet 1 másodperc csupán!

A már letöltött elemek lila színre váltanak és a letölt ikon helyét is átveszi a lila letöltöttséget jelző ikon. 

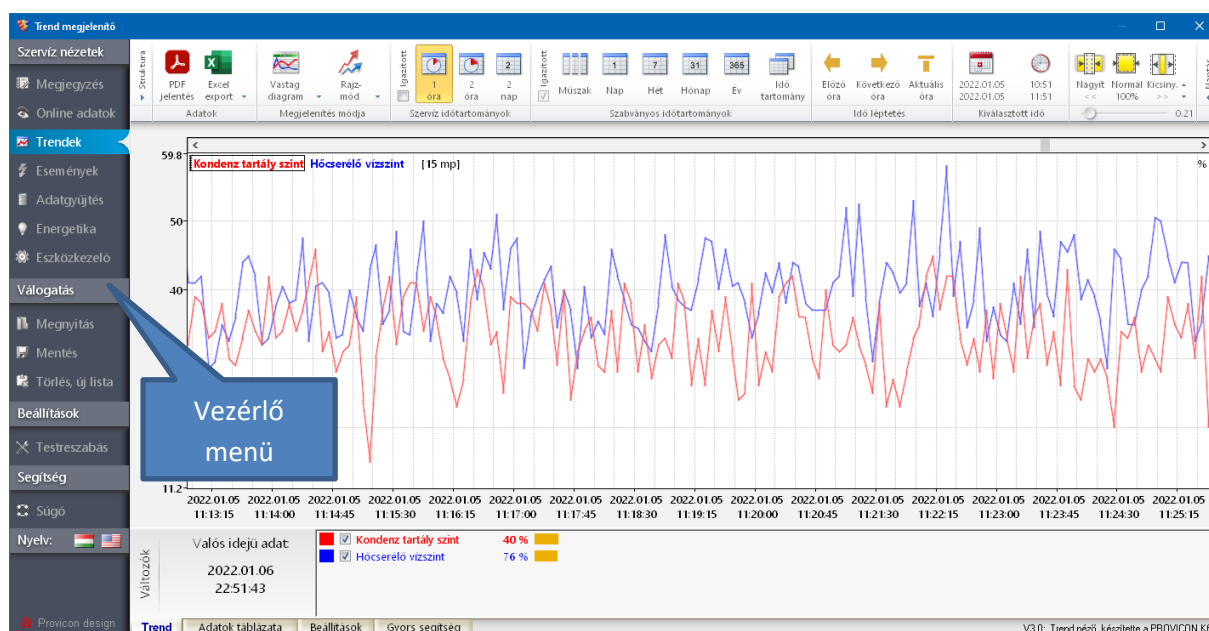
A kiegészítőket bármikor frissíthetjük (frissítés gomb),  sőt törölhetjük is, ha mégsem kívánjuk használni tovább. A program karbantartja a fájlrendszert és a vpr fájlt is megfelelően fogja frissíteni. 

A továbbiakban először a vezérlő menü használatát tisztázzuk, majd összefoglaljuk az egyes kiegészítőket és azok funkcióját.



1.1. Vezérlő menü

Az új kiegészítők mind skálázhatók (nagyíthatók, maximalizálhatók), megőrzik pozíciójukat, és minden kiegészítő elérhető a vezérlő menün keresztül, ami önálló add-on önmagában is. A vezérlő menü egy további kiegészítő tehát, ami az ablakok bal szélén jelenik meg, mint menü. Bármelyik ablakhoz hozzárendelhető:

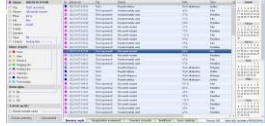


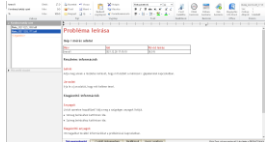
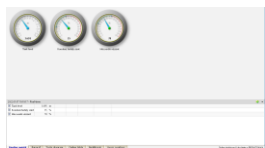
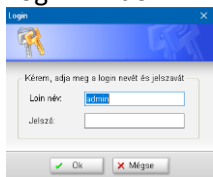
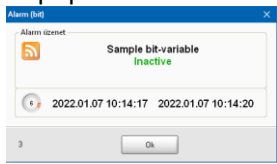
A lényeg az, hogy akármelyik képet rajzoltattuk is fel a programmal, onnan kezdve váltogathatunk a nézetek között: trend, alarm, megjegyzés, adatgyűjtés stb. – mind ugyanazokra a változókra szűrve.

De miért van erre szükség? A magyarázat egészen gyakorlati. Általában úgy használjuk a kiegészítőket, hogy a képen látható objektumokra, gombokra vagy kijelzőre kattintva hívjuk meg, a benne kiválasztott változók adataival. Valójában egy válogatást készítünk, mivel a *Ctrl* gomb nyomvatartásával egyszerre több változót is kijelölhetünk (a fenti trendbe két változót válogattunk össze). Mármost a kérdés az, hogy melyik kiegészítő jelenjen meg a kattintás hatására? A trend? Az alarm? Amennyiben vezérlő menüt használunk, akkor mindig a legutóbbi választás (default = trend). A vezérlő menü használatával végül egyetlen kattintásra vezettük vissza a teljes információs rendszert, beleértve még a CMMS karbantartás menedzsment és energiamenedzsment rendszert is. A program használata ennél egyszerűbb már nem is lehet.

1.2. Kiegészítők összefoglalása

A kiegészítőket az alábbi táblázatban foglaltuk össze:

Kiegészítők	Tipikus ablak	Leírás
Trend	TrendWindow 	Trendek listázása, exportálás Excel-be, csv fájlba, PDF riport készítése. Trend nagyítás, pásztázás, több tucat opció érhető itt el. Bekapcsolhatjuk pl. a min - max állítást, változtathatjuk az időskálát, készíthetünk trendstatistikát stb.
Események, alarm	AlarmWindow 	Események, nyugtázatlan események és fennálló hibák listázása, hibakezelés. Ahogy a trend, az eseménynapló is szelektálható időben, de tetszőleges mezőre, változóra, ill. változókra is. Továbbá SQL query-vel is szűrhető.

Adatgyűjtés 	DataWindow 	Gyűjtött adatok megjelenítése táblázatosan és graikonon. Az adatok exportálhatók Excel-be, csv fájlba, készíthető belőle Excel export, PDF jelentés stb. A megjelenítés vezérlésére itt is létezik több tucat opció, hasonlóan a trendhez.
Megjegyzések	NoteWindow 	A kiegészítővel megjegyzéseket fűzhetünk meglévő alkalmazásaink adataihoz Richtext formátumban (szép, formattált dokumentumok). További szolgáltatás a csatolt dokumentumok rendszere (html, doc, excel, txt, pdf, bmp, jpg, gif, svg...).
Online dashboardok	OnlineWindow 	A kiegészítő alaphelyzetben számlapos analóg kijelzőkön mutatja a változók értékeit, de választható itt oszlopdiagram és tortadiagram nézet is. Letárolhatók továbbá a változók előzményei későbbi összehasonlítás céljából. Megint rengeteg opciónk van a stílus és a megjelenítés állítására.
Karbantartás menedzsment	EquipmentWindow 	E különleges szolgáltatása a programnak egy teljes karbantartás menedzsment rendszer illeszt programunkhoz. Alaphelyzetben a kiválasztott változók vagy név alapján azonosított gépelem adatlapja látszik, de további füleken elérhető a gép életútja, meghibásodásai, karbantartásai stb.
Energiamenedzsment	EMSWindow 	Elérhető egy teljes energiamenedzsment rendszer is a kiegészítő által. Itt valójában egy tallózó ablakot jelenítünk meg, a program pedig az EMS rendszerhez ezen keresztül csatlakozik. Az EMS beállítása egy külön licence keretében, de még mindig VISION alapokon lehetséges.
Beléptetés	LoginWindow 	Ez a kiegészítő csupán egy szép beléptető ablak, amiből könnyen készíthető saját változat. Nagyon fontos azonban, hogy a programot egy VISION-ben rajzolt ablakon vagy képen keresztül léptessük be. Web-en másként nem üzemeltethetünk alkalmazásokat.
Popup alarm	PopupAlarm 	Kiegészítőként valósítottuk meg a popup alarm funkciót is. Ilyet ugyan bárki készíthet megint, de az itt letöltött változat képes megmutatni, hogy az esemény mikor történt, mikor szűnt meg (ha egyáltalán) és mennyi idő telt el a bekövetkezéshez képes, azaz mióta nem nyugtázták.
Egyebek	SetupWindow HelpWindow	A <i>setup</i> képernyőn a kiegészítő testreszabása oldható meg: eldönthetjük, hogy mely kiegészítők jelenjenek meg a vezérlő menüben egyáltalán. A súgó pedig tartalmazza a témáról szóló teljes dokumentációt.

1.3. Vezérlő kiválasztása futó alkalmazásban

Nem elég letölteni a kiegészítőt, azt is meg kell mondanunk, hogyan jelenjen meg és melyik komponense. Ugyanis pár kiegészítő több képből áll. Például a trend esetében választhatunk összetettebb és egyszerűbb trendlistázók közül, eltér ezenkívül a desktop és a webes változat:

VISIONX10		Feldolgozott fájlok listája	Kapcsolódó rendszer funkció cseréje (KY)				Kattintás az objektumon		 
Csomagok (VISION-ben megvalósított...)		Letöltés	Deszktp		Web		Deszktp	Web	Magyarázat
Rendszer adatbázisok	 Trend csomag TrendWindow és tartozékai	  	☒	TrendWindow	☒	wTrendWindowLight	 Trend	 Trend	TrendWindow típusai
	 Alarm csomag AlarmWindow és tartozékai	  	☒	TrendWindow	☒	wAlarmWindow	 Alarm	 Alarm	AlarmWindow típusai
	 Adatgyűjtés csomag DataWindow és tartozékai	  	☒	TrendWindowLight	☒	wDataWindow	 Adat	 Adat	DataWindow típusai
	 Dashboard csomag Online dashboardok és tartozékai	  	☒	TrendWindowWeb	☒	OnlineWindow	 Dash	 Dash	Online adat és dashboard
	 Notepad csomag NoteWindow és tartozékai	  	☒	TrendImage	☒	wNoteWindow	 Note	 Note	Hogyan használjuk a NoteWindow-t?
Extra szolgált.									

Azt dönthetjük el a legördülő listából való elemválasztással, hogy a program KY funkcióhívását mivel fogja helyettesíteni (pl. a **KY 802** hívás attól függően fogja a standard trendlistát mutatni, vagy a legördülő listából választott ablakot, hogy itt mit választunk ki). Mint látható, eldönthetjük ezt külön a desktop alkalmazás és külön a web számára. Nem meglepő, hogy ez utóbbi néha különbözik a vastag kliensen futó alternatívájától, mivel a weben chart elemeket használunk. Általában véve elmondható, hogy a webes változat kevesebbet tud, mint a desktop, ráadásul mások az elvárások is vele szemben. Az ablakok webes megfelelője általában egy 'w' karakterrel kezdődik, de még az is lehet, hogy a vastag kliensen is a webes változatot futtatjuk. Ilyen a trend komponensen belül a *TrendWindowWeb* nevű trendablak. A lényeg tehát, hogy a program az itt kiválasztottnak megfelelően fog automatikusan különbséget tenni a vastag kliensen és a tallózóban futtatott változatok között.

A legördülő lista eléjén látható *checkbox* azt határozza meg, hogy az illető komponens be akarjuk-e ágyazni a **vezérlő menü** belsejébe. Az ablakok ui.  AlarmWindow . A vezérlő menü nélkül és azzal együtt is megjeleníthetők. A vezérlő menü szerepéről az előző fejezetben olvashat többet.

Kattintás az objektumon	
Deszktp	Web
☐ Trend	☐ Trend
☐ Alarm	☐ Alarm
☐ Adat	☐ Adat
☐ Dash	☐ Dash
☐ Note	☐ Note
☐ CMMS	☐ CMMS
☐ EMS	☐ EMS
☒ Menü	☒ Menü

Végül azt kell még meghatároznunk, hogy a képen látható objektumra való **rákattintás** hatására melyik ablak jelenjen meg.

Itt egyértelműen a legelső sor (Menü) a javasolt választás, hisz ezzel érjük el, hogy a vezérlő menüvel dönthessük el, éppen mit látunk.

Egyébként, ha csak az a cél, hogy a változóra kattintva mondjuk a *TrendWindow* jelenjen meg, akkor a legelső sort (Trend) kell kiválasztani.

Vegyük észre, hogy itt is különválasztható a desktop és a web környezet működése. Gyakran választunk vezérlő menüt vastag kliensen, mivel ez a legtöbbet tudó megoldás és csak sima trendet a weben. Hisz a többi kiegészítő elérhető a szokásos módon is – például a sablonban elhelyezett gombok vagy menü objektum (MNU) révén.

Így dől el tehát, hogy a KY rendszerfunkciók és a változó-összeválogatás milyen komponenset hívjon.

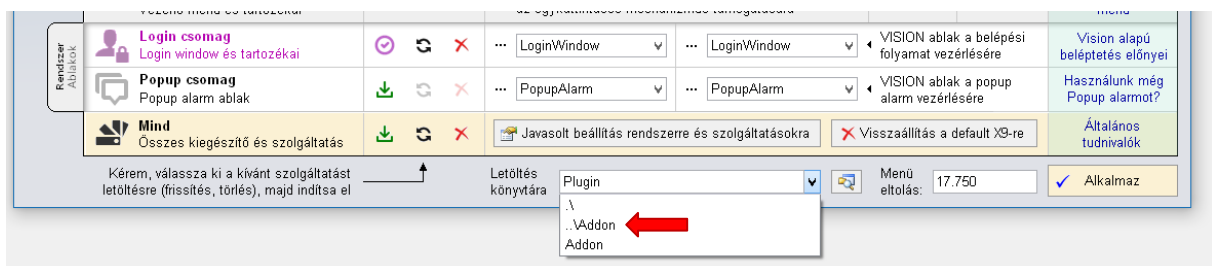
1.4. Miért olyan gyors a kiegészítő letöltése?

Az új kiegészítők nyelvi fordításai külön Excel fájlban vannak **language2.xls** néven. Emiatt nem kell többé összemásolni a szövegeket az alkalmazás saját **language.xls** fájljának tartalmával. Ez tartott a korábbi változatokban sokáig.

A dolog azért érdekel külön figyelmet, mert ezt eljátszhatjuk a saját alkalmazásunk **language.xls** fájljával – abból is lehet mostantól kettő, ha két külön rendszert vegyítünk alkalmazásunkban.

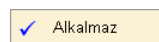
1.5. Összevont alkalmazás közös kiegészítőkkkel

Korábban már utaltam rá, hogy az *Addon* könyvtár megfelelő megadásával elérhetjük, hogy ne az alkalmazás alatt legyenek a kiegészítők, hanem egy könyvtárszinttel feljebb. Például készíthetünk egy közös Addon könyvtárat. Az alkalmazáson belül pedig **..\Addon**-t kell megadni könyvtár gyanánt. Ezt egyébként az *Add-on Manager* fel is kínálja, annyira fontos opció:



De beírhatjuk persze kézzel is. Miután azonban ezt megadtuk, a program a második, harmadik stb. alkalmazás konfigurálása során azt fogja mutatni, hogy a kiegészítők már le vannak töltve. Persze, hisz a **..\Addon** könyvtárban tényleg ott vannak a szükséges fájlok.

Ez azonban még kevés. Meg kell nyomni az **Alkalmaz** gombot is, hogy a szükséges módosításokat a program a **vpr** fájlba is átvezesse – minden alkalmazásra külön-külön.



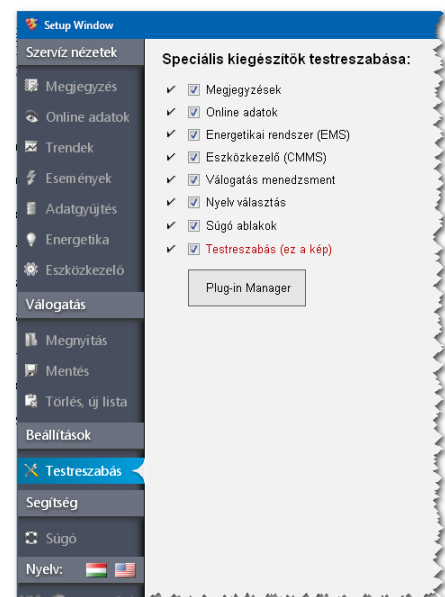
1.6. Vezérlő menü speciális opciói

A vezérlő menünek van két olyan menüpontja, amihez nem tartozik külön kiegészítő; tulajdonképpen magának a vezérlő menü komponensnek a része:

- Súgó
- Setup

Az előbbi az egyszerűbb: egy remek részletes leírás található itt.

A *Setup* képen pedig azt tudjuk eldönteni, hogy a letöltött kiegészítők közül mit jelenítsünk meg ténylegesen is. Ez a **testreszabás**. Egyszerűen vegyük ki azokat a pipákat, amiket nem szeretnénk a vezérlő menü keresztül elérni. Kapásból magát a Testreszabás képet is levehetjük, miután beállítottuk, hogy a kezelő ne tudja később átkonfigurálni (konfig jog kell).



Vegyük észre, hogy az Add-on Manager is elérhető innen, azonban már csak azzal a céllal, hogy az egyes kiegészítők desktop és web alternatíváját megváltoztassuk. A letöltés ilyenkor már tiltott!

1.7. Saját Add-on menedzser

Az előző fejezetben bemutatott lehetőség vezet el a saját Add-on menedzser gondolatához. Előfordulhat, hogy valaki olyan alkalmazást szeretne készíteni, ahol maga a felhasználó választ a különféle egyszerűbb és bonyolultabb kiegészítők közül, hisz ez részben ízlés kérdése, és emeli az alkalmazást. De nyilván nem akarjuk felkínálni neki az összes lehetőséget. Hogyan oldható ez meg?

Azt kell tudni, hogy létezik egy **Assignments.xdb** nevű XDB adatbázis fájl az alkalmazás könyvtára felett – ugyanott, ahol a **Settings.xdb** (a kiegészítők belső konfigurációs adatbázisa) is megtalálható. Ezt használja a program. A fájl felépítése nagyon egyszerű és önmagáért beszél:

Trend	Alarm	Data	Online	Note	CMMS	EMS	Login	Popup	Click	ClickWindow
TrendWindow	AlarmWindow	DataWindow	OnlineWindow	NoteWindow	EquipmentWindow	EMSWindow	LoginWindow	PopupAlarm	7	Service
wTrendWindowLight	wAlarmWindow	DataWindow	OnlineWindow	wNoteWindow	EquipmentWindow	EMSWindow	LoginWindow	PopupAlarm	7	Service

Ha például a TrendWindow desktop váltzatát szeretnénk módosítani, ezt az utasítást kell kiadni az alkalmazás programjából:

```
xdbWrite 1, "Trend", "myTrendWindow", "..\Assignments";
```

A legelső paraméter felel meg annak, hogy a vastag klienst vagy a webet módosítottuk (1: desktop; 2: web), a második paraméter az oszlop neve (itt: *Trend*), a harmadik pedig a kiegészítő neve, amivel a korábbi szeretnénk felülni (itt: *myTrendWindow*).

A többi oszlop rendre megfelel az egyes kiegészítőknek. A legutolsó oszlop (*ClickWindow*) pedig azt a kiegészítőt határozza meg, amelyiket rákattintás hatására szeretnénk megjeleníteni. A **Service** felel meg annak, hogy a vezérlő menüt hívjuk meg. Ha azonban csak a TrendWindow-t akarjuk használni, írjuk be ide: `xdbWrite 1, "ClickWindow", "TrendWindow", "..\Assignments";`

Természetesen az Assignments fájl "kézzel", mondjuk Notepad-del is könnyen szerkeszthető.

Ugyanez a gondolatmenet vezet el végül a saját kiegészítők rendszeréig, amit pl. a rendszer saját kiegészítőiből magunk fejlesztettünk...

2. Változók összeválogatása képről web-en

Ez a funkció eddig desktop alkalmazásból működött csak, mostantól azonban web-en is. Vagyis a *Ctrl* és *Shift* gombokat használhatjuk ezentúl tallózóban is. De természetesen csak azokon a eszközökön, ahol ez egyáltalán lehetséges (pl. iPad, iPhone, Android stb. eszközökön továbbra sem lehetséges, mivel nincs kontroll billentyűjük – ott marad az egykattintásos módszer). A szokásos, elterjedt tallózókban azonban működik szépen. Működik még az időközben kihalt Internet Exploreren is:

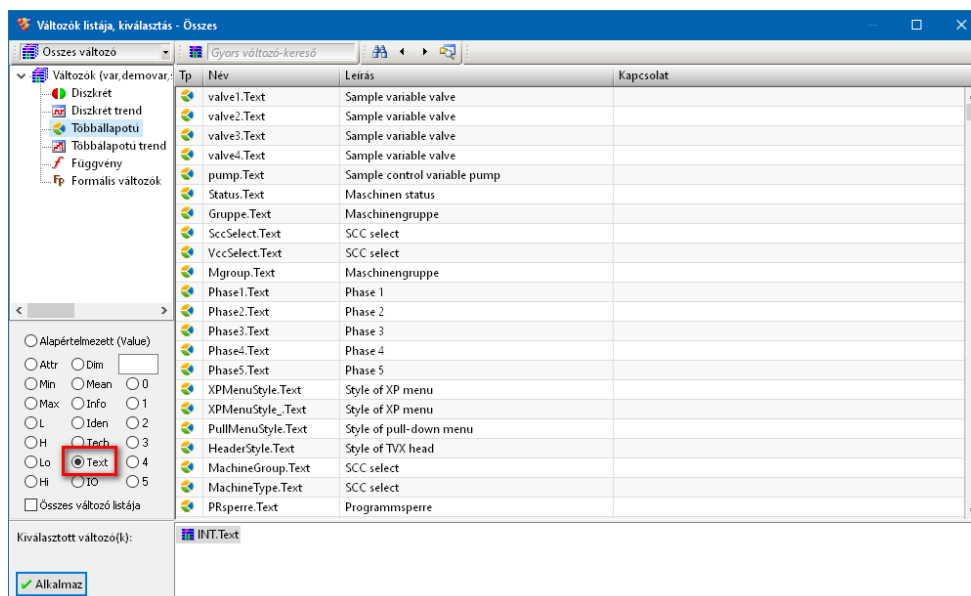


Emlékeztetőül, korábban ezt üzemmódváltással oldottuk meg úgy, hogy az alkalmazás trend-összeválogatás üzemmódba kapcsoltuk, majd a végén vissza alapállapotba. És e legutolsó művelet hatására jelentettük meg a *TrendWindow*-t. Az új változatban azonban elég csak felengedni a *Ctrl* vagy *Shift* gombokat a tallózóban, és ha előtte összeválogattunk változókat, akkor azok egyszerre jelennek meg a felugró ablakban. Mintha csak vastag kliensen végeztük volna az összeválogatást.

X10 Verzió 2.27

1. Fontos változtatások

- A módosítás érzékeny kommunikáció web kliensből eddig nem működött. Az új verzióval ez is lehetségessé vált az összes releváns objektumból: editbox, checkbox, radio button, drop-down list és persze értékadás esemény kiszolgálóban.
- Javítások történtek a többmonitoros üzemmód 2.26-os verziójában bevezetett megoldáshoz képest.
- Formális változók típusjegyzetése az összes lehetséges kombinációra.
- Bevezetésre került a *Text* változóreferencia a változók listájában:



Ez a diszkrét és a többállapotú változók egyik fontos attribútuma, amikor a változóhoz tartozó szöveges információt akarjuk kiírni.

- Sebesség mérés kiterjesztése 16 magig.
- WebCommand parancs kiegészítése HTTPS képességgel (SSL/TLS).
- SMS és SMS2 driverek "csendes" üzemmódjában történtek további javítások.
- Történt néhány további apró hibajavítás (nem implementált SVG objektumok eltűntetése pl. az objektum választóból)

2. 64-bites Microsoft Access

Többen jelezték, hogy problémájuk volt a 64-bites Access ODBC driver Microsoft honlapjáról való letöltésével. Ennek az az oka, hogy a Microsoft az új verzióban egyeztet a gépen található Office-szal, és ha az 32-bites, nem hajlandó 64-bites ODBC-t telepíteni (sőt office nélkül sem). "Okos". Ugyanakkor az MS honlapon megtalálható a régi, jó megoldás is, csak keresni kell:

<https://www.microsoft.com/en-us/download/details.aspx?id=13255>

De innen is letölthető:

http://www.provicon.net:8000/Vision/AccessDatabaseEngine_X64.exe

Ezzel lehet tehát a 64-bites Access hibáját maradéktalanul kijavítani!

A másik változtatás, hogy amennyiben 64-biten nem tudja a program megnyitni az MS-Access alarm adatbázist, egy weboldalt is megnyit, ahol a szükséges teendőket részletesen ismerteti:

Microsoft Access 64-bites adatbázis kezelő

Jelenleg az Access 64-bites változata nem elérhető az Ön számítógépén! Az alábbi linken keresztül azonban letöltheti és onnan kezdve 32- és 64-bit alatt egyaránt működni fog.

A probléma háttere röviden a következő:

A 32-bites **MS-Access provider** (Microsoft Jet 4.0...) és **ODBC driver** minden Windows operációs rendszernek automatikusan része. Nem kell hozzá feltelepíteni semmit. Azonban a 64-bites ODBC drivert sajnos a 64-bites Windows sem tartalmazza! 😞 Két dolgot lehet tenni:

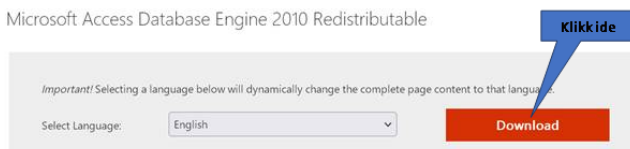
- 64-bites Office Pro-t telepíteni (Office változat, aminek az Access is része)
- Az alábbi linken található adatbázis motort kell feltelepíteni

Tudni kell még, hogy bár a VISION natív drivere tartalmazza az MS-Access-t, az csak a providert jelenti (korszerű OLE-DB alapú adatbázis kezelők az adatbázist providerekkel keresztül érik el). Azonban az Access provider az ODBC alapdrivert hívja, amint egyébként az eredeti Access provider is. A 64-bites ODBC driverre tehát mindenképp szükség van.

Az adatbázis motor letöltése az alábbi linkről lehetséges:

<https://www.microsoft.com/en-us/download/details.aspx?id=13255>

Microsoft Access Database Engine 2010 Redistributable

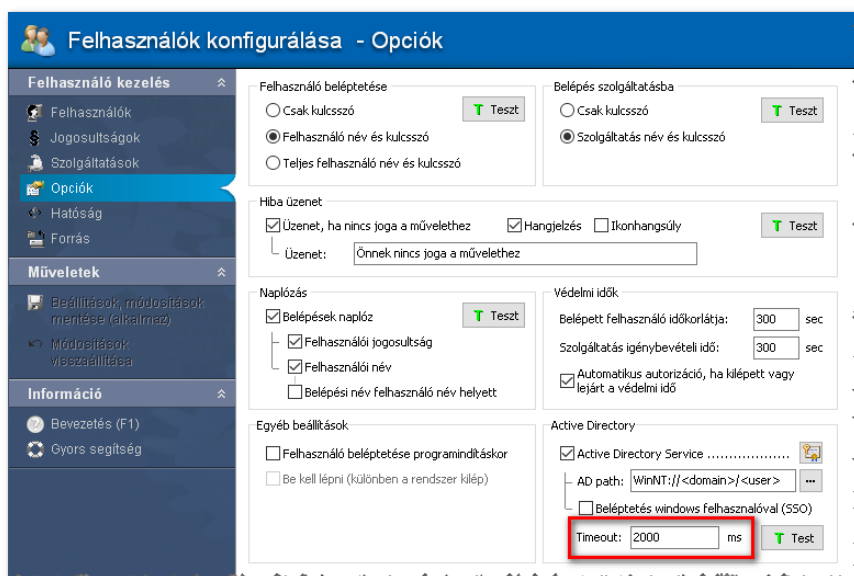


3. Active Directory időkorlát

A Microsoft Active Directory felhasználó kezelésével az egyik fő probléma az, hogy nagyon sokáig tart a hibás felhasználó név és jelszó megállapítása. Van olyan rendszer, ahol 40 másodpercig!

E Microsoft probléma kiküszöbölésére készült egy megoldás: a beléptetést önálló szálon indítjuk el és saját időkorláttal termináljuk. Abból indulunk ki, hogy amennyiben a beléptetés tovább tart, mint mondjuk 2 másodperc, akkor az már biztosan sikertelen, csak még ki kell várni a végét (20-40 sec).

Az időkorlát beállítása a felhasználók, opciók menüpontban lehetséges:



Figyelem: csak SQL-be, vagy XDB (SQL)-be konvertált felhasználói adatbázis esetén működik. A bináris USER.DAT beállítással nem, mivel abban nem lehet új mezőket elhelyezni.

X10 Verzió 2.26

1. CMMS alkalmazás beillesztése

Pár verzióval korábban beszámoltunk róla, hogy elkészült a CMMS (Computerized Maintenance Management System) teljesen VISION X10 alapú változata. A megoldás számos előnnyel jár:

- Közvetlenül beépíthető bármilyen meglévő alkalmazásba
- Web képes, azaz internet böngészőben is futtatható
- Jóval gyorsabb, látványosabb és átalakítható, ha szükséges

A beépítést *szuperprojekt* készítésével kell elvégezni a szokásos módon.

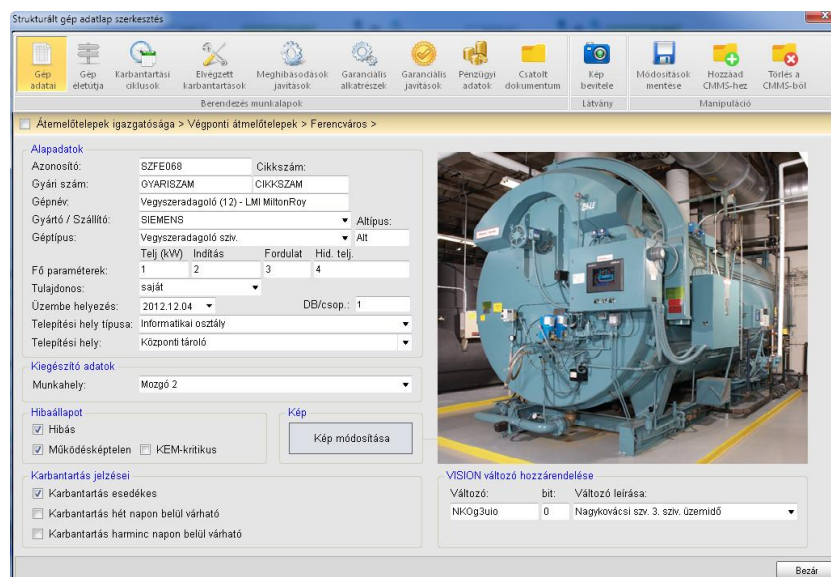
1. Fel kell venni a visCMMS programot az VISION projekt listába (VPL)
2. Majd kibővíteni vagy elkészíteni a szuperprojektet, aminek a visCMMS is része

```
Project begin
  Title=           "RG_EK alkalmazás"
  Icon=            "bin\icon\squares.bmp"
  CommandLine=     "RG_Main, RG_Gkazan, RG_Kiskazanok, KKI_Vacuum, visCMMS"
  X10FileSystem=   Yes
  XOffComm=        Yes
End
```

Ezután az alkalmazás menürendszerét érdemes átalakítani úgy, hogy a visCMMS képeire hivatkozni lehessen. A főkép neve: **CMMS**.

Van azonban egy ennél fontosabb kép, amire ablakrajzolással hivatkozhatunk. Ennek **Machine** a neve, paramétere pedig a megjeleníteni kívánt gép azonosítója (KKS kódja). Abban az esetben, ha már egyébként is használjuk a *TrendWindow*, *AlarmWindow*, *ServiceWindow* stb. szolgáltatásokat, az *OnControl()* hívásba érdemes beépíteni. Például úgy, hogy egy hyperlinket készítünk és az *OnControl*-nak a *Caption*-t adjuk át. Ahogy jobb oldalt látszik. Ezután az SZFE068-nak megfelelő gépelem már rákattintással megjeleníthető és abban a CMMS leírásában leírt számtalan szolgáltatást máris használhatjuk.

```
TX8:TX  ".SZFE068"
BEGIN
  Position=      32.625, 62.5
  Font=          "Arial", 12, Bold, 0
  Justification= Right, Bottom, False
  Shadow=        False
  EventRefresh=  erPoint
  Cursor=        erPoint
  Click          OnControl Caption
```



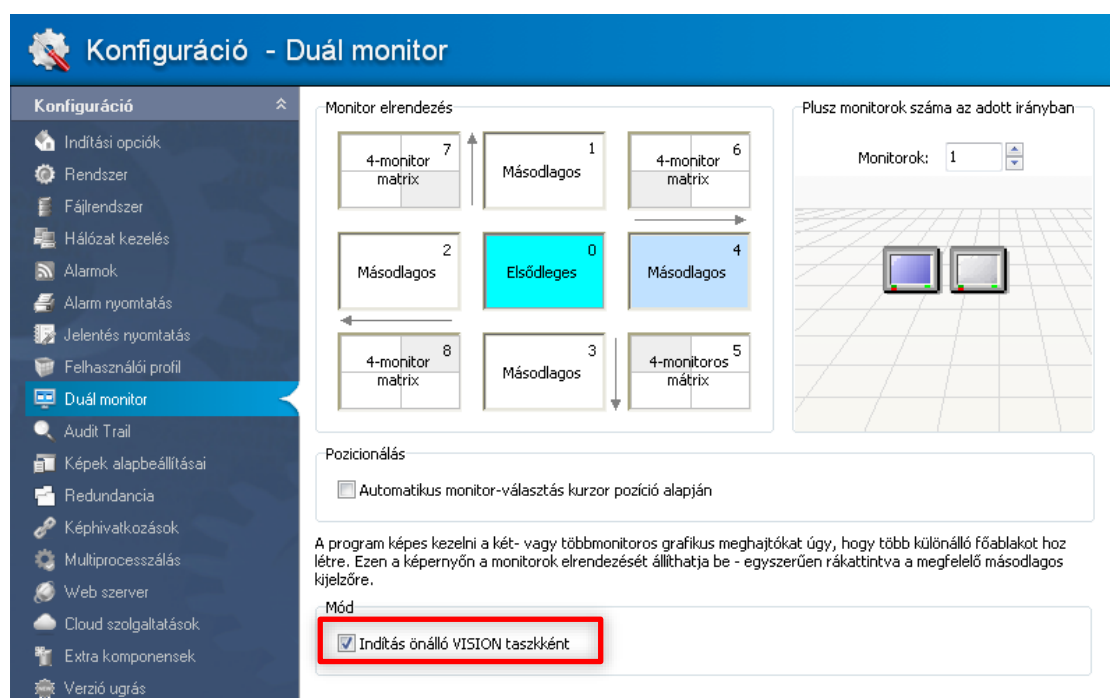
2. Többmonitoros rendszerek új lehetőségei

Mint ismeretes, a VISION az alkalmazások futtatása során képes akár négy monitorra is egymástól független képet rajzolni különféle monitor elrendezésekben (vízszintesen, függőlegesen, mátrixban).

E szolgáltatás új opcióval bővült: mostantól önálló VISION tasztként is elindítható a 2-ik, 3-ik stb. monitoron látható példány, ami hálózati kliensként kapcsolódik azután a rendszerbe. A megoldás legfontosabb előnye, hogy a hálózati átvitelből kitiltott változók (*Public*) így tényleg eltérő állapotot mutatnak az egyes monitor példányokon. Például abban az esetben, ha a sablonon megjelenítettük az épp látható kép nevét, vagy egy ikont színeztünk vele!

A korábbi megoldásban ez nem volt lehetséges, mivel a VISION csak egy új ablakot nyitott a további monitorokon – értelemszerűen ugyanazokat a változó értékeket / állapotokat mutatva a *Public* változókra nézve is.

Az új opció a *Duál monitor* beállításánál található, alul:



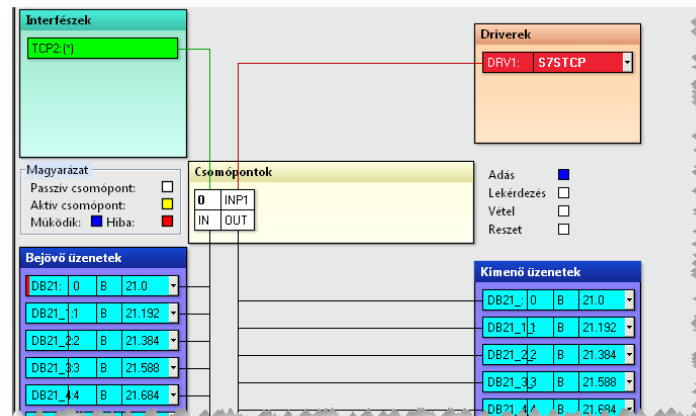
Az indítás és a leállítás is automatikus, mintha csak a korábbi megoldást használnánk.

További előny, hogy a közös szolgáltatások, mint az alarm- vagy a trendablak is több példányban elérhető így minden monitoron, mivel mindegyiken egy teljes értékű VISION példány fut. A megoldás hátránya, hogy valamivel több erőforrás (memória) és hálózati licence szükséges hozzá és külön-külön be kell mindegyiken jelentkezni.

(A többmonitoros rendszerek nem tévesztendő össze a mátrix-monitoros megjelenítéssel. Ez utóbbi esetben egyetlen nagyfelbontású képet rajzolunk a 2x2-es, 2x4-es stb. monitor falra, míg az előbbi esetben több, különböző képet.)

3. Siemens S7 slave teszt driver

A rendszer egy S7 slave driverrel bővül szimulációs célból, hogy a népeszrű S7TCP drivert offline / teszt környezetben is ki lehessen próbálni – PLC nélkül. Az új driver S7STCP nevet kapott. Konfigurációja a MODSTCP, Modbus TPC slave driverhez hasonló, azaz fel kell sorolni a kimeneteket is és a bemeneteket is a csomópont két oldalán. Ezzel azt érjük el, hogy az S7STCP-t automatikusan felkonfiguráljuk a megadott DB és DW offset kombinációkra. Következzen egy példa:



Itt a **B** típust láthatjuk, de **D**-re is így kell beállítani a programot. Általában két ilyen taszkot kell felkonfigurálnunk, mivel a tipikus S7TCP kommunikáció két kapcsolatot, két portot használ: 2000, 2001. Mindkét esetben meg kell ismételniük a kimenetet a bemenet oldalán és megfordítva.

A tesztkörnyezetet a legegyszerűbben úgy tudjuk létrehozni, hogy egyszerűen lemásoljuk a meglévő S7 kommunikációs taszkunkat, abban kicseréljük a drivert (S7TCP -> S7STCP), átírjuk a driver üzemmódját masterről slaverre, és a csomópontban az IP címet '*'-ra (csillagra) cseréljük. Mint a MODSTCP driver esetén. Végül megismételjük a bemeneteket a kimenetek oldalán. Majd az eljárást elvégezzük a 2001-es portra is egy külön taszkban.

A slave drivert ugyanabban a VISION alkalmazásban futtatjuk, de egy külön gépen. Ezáltal a VISION változók ugyanazok lesznek, amik előtte voltak a memóriában, hisz a slave azt adja vissza a master S7TCP szervernek. Nem nullákat, hanem valódi adatokat kapunk tehát a teszt PLC-ből.

4. Demo üzemmód

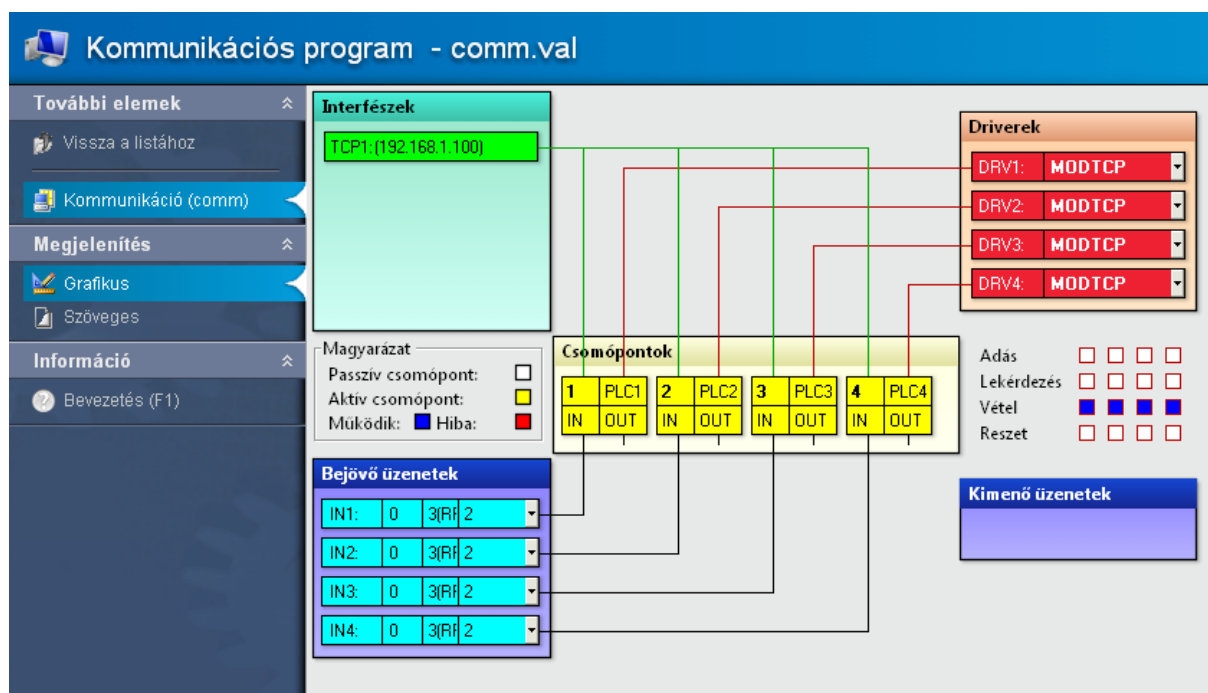
A teszteléshez kapcsolódik a Demo üzemmód is. Ez egy régi funkció, de változtatni kellett az egyik gyorsító billentyű kombináción, nevezetesen a futtató – fejlesztő üzemmód kapcsolóján (új kódja: Ctrl-Alt-O), hogy visszakapjuk a felvétel indítása parancsot. A demó parancsok a következők:

Ctrl-Alt-D	Demó mód ki-bekapcsolása
Ctrl-Alt-R	Felvétel indítása (record)
Ctrl-Alt-S	Felvétel leállítása (stop)
Ctrl-Alt-P	Felvétel lejátszása (play)
Ctrl-Alt-Q	Felvétel ciklikus visszajátszása (cyclic replay)
Ctrl-Alt-M	Felvétel mentése .dmo fájlba (felvétel közben kiadva leállítja és menti a felvételt, egyébként visszaolvassa)

X10 Verzió 2.25

1. Modbus TCP kapcsolatok korlátozása (MOXA)

A TCP/IP kommunikációt érdemes annyi taszkra bontani, ahány csomópont van, mivel az adja a leggyorsabb és legstabilabb eredményt, lévén a csomópontok egymástól függetlenül működnek. Ha pl. valamelyik ág timeout-ra fut, a többi ez egyáltalán nem befolyásolja. A párhuzamosítás megvalósítása VISION-ben úgy történik, hogy annyi driver objektumot veszünk fel (piros), ahány csomópont van, és az egyes csomópontokat külön-külön a driverekhez hozzárendeljük:



Ennek a megoldásnak azonban hátránya, hogy annyi kapcsolat is épül fel, ahány driverünk van!

Hozzárendelés	
☐ Driver név	MODTCP
☐ Driver mód	comMaster
☐ Saját cím	1
☐ Timeout	5000
☐ Várakozás küldés	1000
☐ Várakozás reszt	0
☒ Státusz	dsStandby
☐ Engedélyezett	☒ True
☐ Paraméter lista	false,false,true,false
☐ DontKeepAlive	false
☐ ConnectAway	false
☒ ConnectByChannel	true
☐ PingTest	false

Márpedig ez bizonyos esetekben problémát okoz. Például MOXA használata esetén, ami a kapcsolatok számát típustól függően 4-, 16- vagy 32-re korlátozza. Eszközünk azonban lehet ennél jóval több.

A fent említett probléma megoldására vezettük be a Modbus TCP driverhez a *ConnectByChannel* paramétert: amennyiben ez *True* (1), akkor a kapcsolatot nem csomópontonként, hanem csatornánként valósítja meg a program, ami az interfész objektummal azonosított (zöld). A többi paraméter jelentése a teljesség kedvéért:

DontKeepAlive

A kapcsolatot a driver nem tartja fenn, ha a másik fél (PLC, gateway) lekapcsolta azt (ez a ritkán lekérdezett GPS eszközök esete)

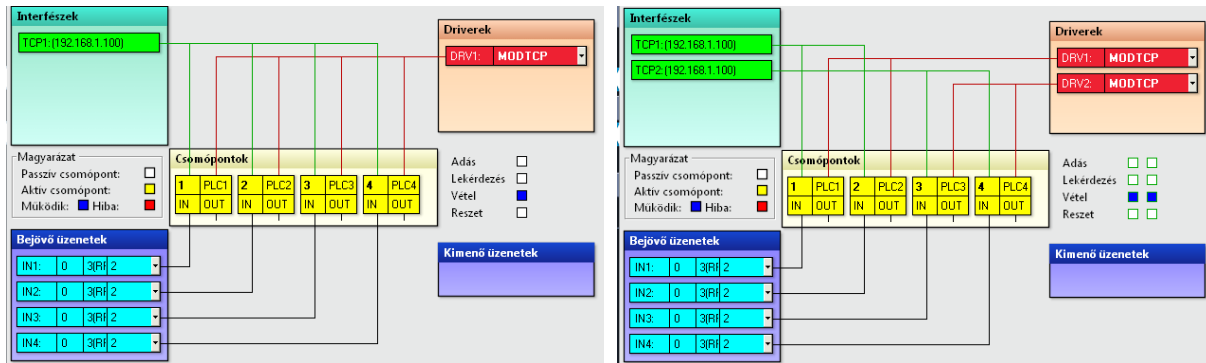
ConnectAway

Azonnali kapcsolatfelvételvételi vagy adási szándék esetén. Egyébként a program a kapcsolatfelvételre egy háttér szálat indít, hogy ezzel a fő szálat ne terhelje (*DontKeepAlive*-val együtt szokás használni)

PingTest

Kapcsolatfelvétel előtt az IP címet a program *ping*-eli. Ezzel offline környezetben segíthetjük az alkalmazásunk futtatását, mivel a kapcsolatfelvételt a program el sem indítja, ha az IP nem ping-elhető

Visszatérve a kapcsolat-korlátozás problémájához, nézzük most meg, hogyan kell beállítani a kommunikációt abban az esetben, ha csak egy kapcsolatot szeretnénk felépíteni, és mit kell tenni akkor, ha kettőt (balra: 1 kapcsolat; jobbra: 2 kapcsolat):



A *ConnectByChannel* paraméter mindkét esetben *True*.

Vegyük észre, hogy két interfész objektumhoz két driver dukál, öthöz öt, stb., hiszen ha eleve két kapcsolatot építünk fel, miért ne futtatnánk azokat külön taszkként!?

A lényeg, hogy ebben az esetben mi szabályozhatjuk, hogy x csomópont hány kapcsolaton keresztül épüljön fel. Csinálhatunk pl. 4 kapcsolatot 32 eszközre, vagy 16 kapcsolatot 32 eszközre – az alkalmazott gateway (MOXA) kapacitásától függően.

Vegyük a döntésnél figyelembe, hogy ha elhasználtuk az összes lehetséges kapcsolatot, más kommunikáció már nem indítható, ill. egymástól átvehetik így a kontrollt. Azaz hagyjunk tartalékot!

Más részről a kapcsolatok számát a bevezetőben említettek miatt érdemes a lehető legnagyobbra választani. Kompromisszumot kell tehát kötnünk az eszköz kapacitáskorlátja miatt.

2. Módosított kommunikációs driverek

A következő driverek módosultak:

1. Modtcp: A módosítás oka az előző fejezetben olvasható
2. Modstcp: A slave Modbus TCP driver kapcsolat-számát 64-re emeltük
3. S7TCP: A biztonságosabb újrakonnektálás érdekében történtek változtatások
4. SMS, SMS2: Bevezetésre került egy AT parancs-blokkoló paraméter

Hozzárendelés	
Driver név	SMS
Driver mód	comMaster
Saját cím	1
Timeout	5000
Várakozás küldés	1000
Várakozás reset	0
Státusz	dsStandby
Engedélyezett	☒ True
Paraméter lista	True
BlockAT	True

A 4. módosítás arra szolgál, hogy ritka kommunikáció esetén feleslegesen ne terhelje a vonalat a folyamatos AT parancs küldés (ami egyébként arra szolgálna, hogy az SMS modem rendelkezésre állását tesztelje). A blokkoló paraméter neve: BlockAT.

3. Ping teszt rendszer szintű engedélyezése

A 2.14-es verzióval bekerült a rendszerbe egy driver szintű változtatás: *ping* teszt. Ennek az volt a fő oka, hogy offline környezetben – ahol nem áll rendelkezésre a teljes kommunikációs környezet – is lehessen futtatni az alkalmazást anélkül, hogy az folyamatosan több tucat vagy több száz végponthoz

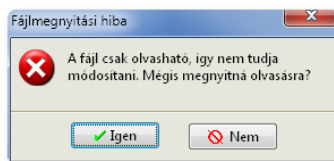
próbálna csatlakozni. Pingelést használva időközönként megnézi, hogy az adott IP létezik-e, és ha létezik, kapcsolódik. Különben nem.

Mint kiderült, vannak azonban olyan alkalmazások, ahol nagyon sok végpont van és sok közülük nem pingelhető, vagy a 200ms-os ping timeout kevés. Ezenél nagyon sokat kellene konfigurálni ahhoz (pingtest paraméter kivétele a drivernél), hogy a korábbi működés visszaálljon. Ezért a ping mostantól generálisan is tiltható/engedélyezhető a vpr.ini fájlban (Bin alatt):

```
PingTest= 0
```

4. Írásvédettség jelzése

Az alkalmazások futtathatóak csak olvasható fájlrendszeren is, de módosítani a fájlokat ilyen esetben természetesen már nem lehet. Egy változtatás mostantól külön felhívja erre a figyelmet:



És ha ennek ellenére mégis megnyitjuk olvasásra a képet, akkor a szerkesztő ablak háttérének az elszínezésével a program jelzi, hogy bár láthatjuk és módosíthatjuk is a szerkesztőablak tartalmát, azt nem menthetjük el:

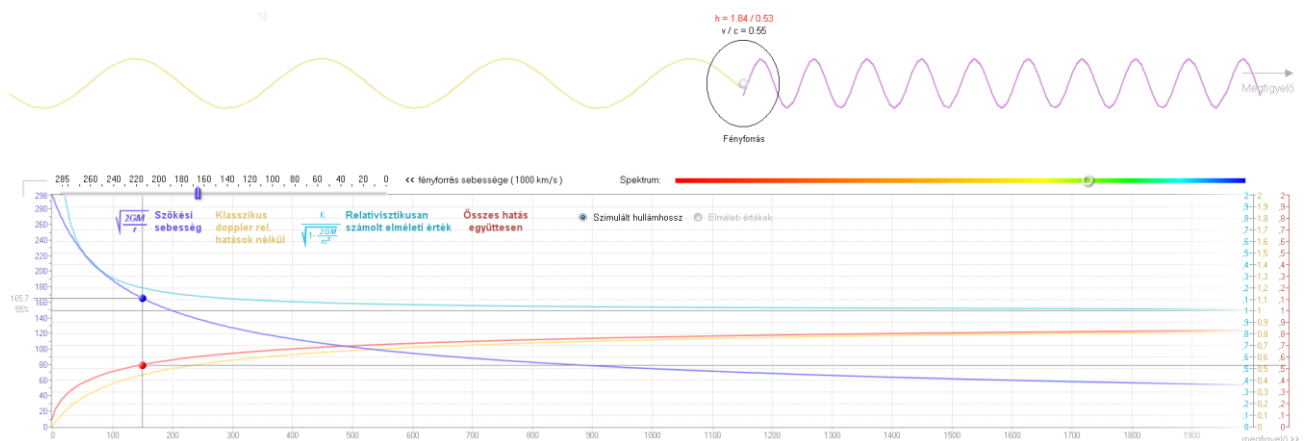
```
1 #fun IMG
2 #pub Global
3 #lib Library
4
5 - ExcelView: Image
6 BEGIN
7   Title= "Excel megjelenítése VISION-nel Office nélkül"
8   Scale= 0,0,PrjWidth/8,PrjHeight/8
9   Color= 124
10  BkgndColor= 125
11  ImageFill= SolidFill
```

5. Új scripting programozási példák

A SAT alkalmazás számos új tudományos scripting programozási példával bővült. Részben a VISION X10 funkcionalitását és sebességét szemléltetendő, részben a tudományos érdekességek iránt érdeklődők kedvéért:

- Fényelhajlás okának magyarázata (miért halad lassabban a fény az üvegben)
- Vöröseltolódás (doppler) jelensége

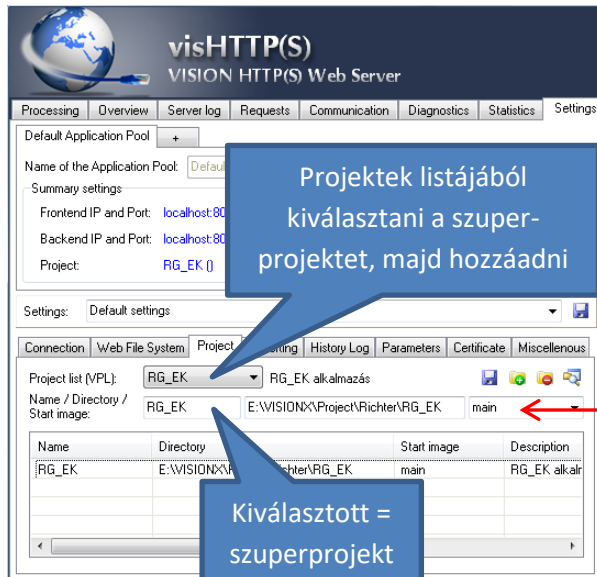
Ez utóbbi három képet érint: *MovingSource*, *StretchedSpace*, *MovingSpace*. Érdekessége, hogy ezekkel az egyszerű VISION szimulációkkal be lehet bizonyítani még az univerzum tágulását, sőt gyorsuló ütemű tágulását is. Az anyaghoz [részletes leírás](#) is készült.



X10 Verzió 2.24

1. Összevont alkalmazás beállítása a Frontend-ben

Felvetődik a kérdés, hogy ha nem 1 alkalmazást kívánunk elérhetővé tenni web-en, hanem több projektet összevonva, azaz szuperalkalmazást, akkor mi a teendő? A válasz: ilyenkor magát a szuperprojektet adjuk meg. Nem szükséges tehát felvenni a szuperprojekt összetevőit. Ha mégis felvettük, akkor egy dupla kattintással válasszuk ki a szuperprojektet a listából, ahogy alább látszik:



Amikor tehát regisztráljuk, elég magát a szuperprojektet kiválasztani (képen: RG_EK), azt hozzáadni a projektek listájához (legalul), majd megadni a kezdőképet (képen: main).

Ez utóbbi művelet nem feltétlenül szükséges, de a kezdőképen (index.html) a program igényelheti az induló kép nevét, ami azonban nem azonos a szuperprojekt kezdőképével, lévén ez utóbbinak nincs is ilyen. Ezért a kezdőképet kézzel kell megadni. Mégpedig a főalkalmazás kezdőképét, amivel majd az egész webalkalmazás indul. De még egyszer, ez nem feltétlenül szükséges, a működést nem befolyásolja. Az index fájlt egyébként is célszerű a felhasználóra szabni, vagy

abban automatikusan a kezdőképre váltani (ld. alább), ott pedig a felhasználót a *LoginWindow* képpel vagy hasonlóval beléptetni! Ezek nagyon fontos előkészítő műveletek!

A program induláskor elolvassa a szuperalkalmazás vpr fájljának *CommandLine* bejegyzését és a benne található összes projektet automatikusan regisztrálja. Sőt ki is keresi közülük a főalkalmazást, hogy egyéb megadás híján azt kívánja fel az index oldalon (ide kellett a kezdőkép):



Innen kezdve bármelyik projekt bármelyik képére hivatkozhatunk: <projekt>/<képnév>.

Abban az esetben, ha ezt kikerülnénk – mindjárt a főalkalmazás kezdőképére váltva –, akkor tegyük a következőt:

1. A Web\Root\Hun alatt az *custom_.html* fájlt nevezzük át *custom.html*-nek. Ezzel elértük, hogy a program default hivatkozása ne az *index.html* legyen, hanem a *custom.html*
2. Készítsünk felhasználóra szabott oldalt vagy írjuk be a *custom.html* fájl <body> sorába a következőt – persze a saját neveinkkel:

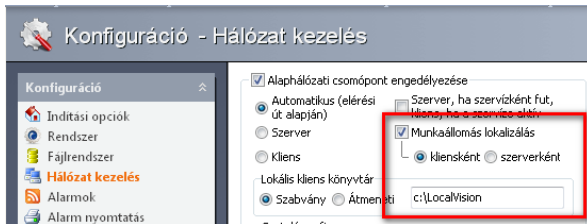
```
<body topmargin="0" leftmargin="0" rightmargin="0" ...
onLoad="location.href='RG_Main/main'">
```

A szuperprojekt csak akkor működik web-en, ha az összes projekt szerepel a projekt listában!

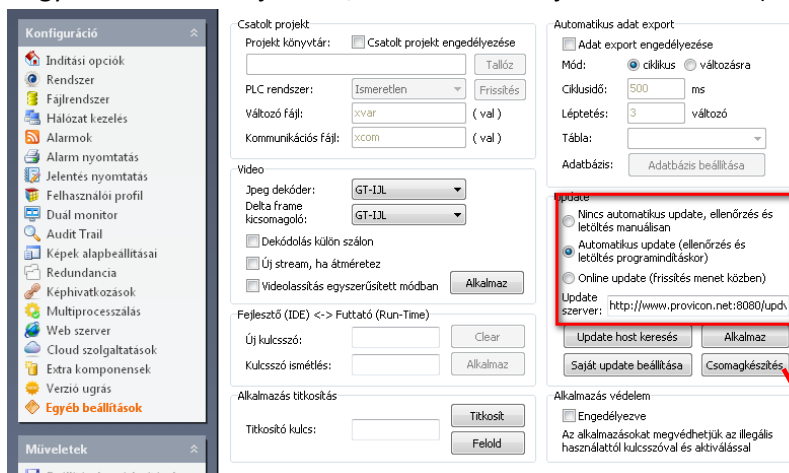
2. Lokalizált cloud elérésű környezet kialakítása

A szerver oldali konfiguráció során arra kell ügyelni, hogy a szinkronizáció forrásának IP címe ne legyen azonos a kliens IP címével, vagy legalább ne közvetben hivatkozzunk rá. Azt ui. a program figyeli, hogy ugyanazon a gépen vagyunk-e, mint a szerver – azt elkerülendő, hogy a szerver maga is szinkronizálja a saját fájljait önmagára. Egy tanfolyamon rontottuk ezt el, tanulságos volt és szükséges külön felhívni rá a figyelmet. A végrehajtandó lépések a következők:

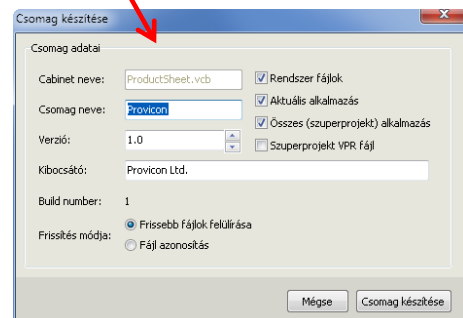
1. Lokalizált kliens üzemmód beállítása, emiatt fogja elkészíteni a szükséges fájlok másolatát a program egy `c:\LocalVision` könyvtárban (lehetne máshol is):



2. Egyéb beállításoknál az update mód kiválasztása és a cloud host beállítása – itt kell ügyelni arra, hogy valós IP címet írjunk be, amit a kliens majd kívülről is elér (saját IP nem jó!):



3. El kell készíteni a programról egy csomagot, ami a szinkronizációhoz használt 1 db cabinet fájl lesz (ProductSheet.vcb). Töltsük ki megfelelően, mivel a kliens programból ezek az információk lekérdezhetők. Kiiratható pl. a saját- és a forrás verziószáma, amit itt adunk meg.



4. Más csak a kliens oldali indító ikon hiányzik, aminek az első indítás során a szerver programra, mint osztott könyvtárra kell hivatkoznia. Például:

`\\server\E\Packages\VisionX10\Files\Bin\_vpr.exe -Alpine`

5. Indítsuk ezt el és figyeljük meg, ahogy létrejön a `c:\LocalVision` fájlrendszer a gépen.

6. Miután ez megtörtént, a programot már akár lokálisan is indíthatjuk. A program mindig szinkronizálni fogja úgy az alkalmazást, mint a frissített VISION-t. És a megosztást meg is szüntethetjük. Az egészről videó is készült:

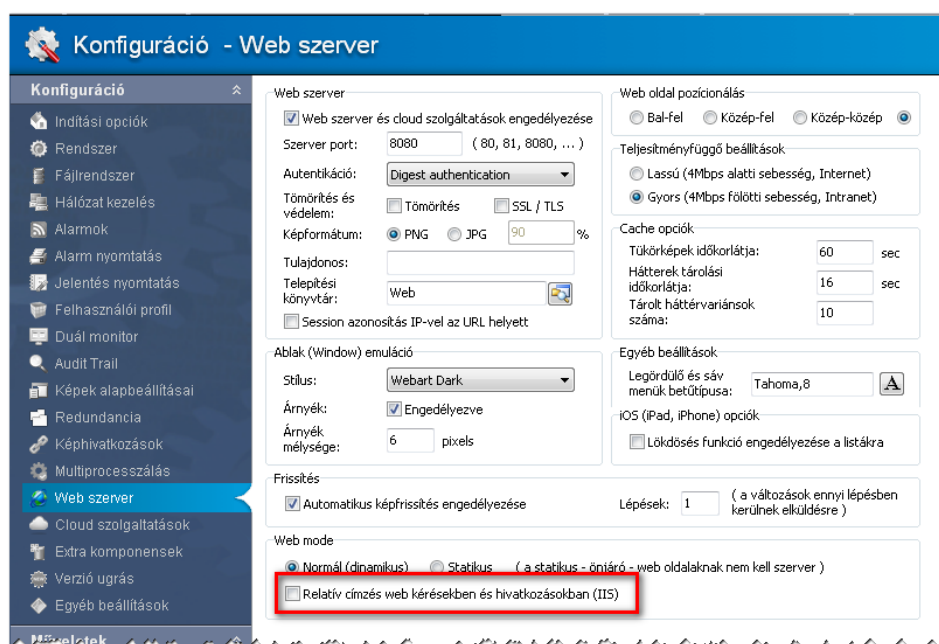
<http://www.provicon.net:8000/video/Education/VisionCloudLocalClient.html>

3. Relatív elérésű web-es környezet

Ez a talányos cím az IIS (Internet Information Services) környezetbe ágyazott VISION web alkalmazásokra utal. Előfordulhat ugyanis, hogy a VISION-nel fejlesztett web alkalmazást nem önálló URL alatt kívánják elérni, hanem egy magasabb szintű vállalati rendszerbe ágyazottan. Erre a megoldásra egyre növekvő az igény, **Azure környezetben is ez a megoldás.** S mivel a VISION-nek saját web szervere van (Vision, visHTTPS), ezért a kérdés úgy merül fel, hogy hozzáfűzhető-e az IIS a VISION web szerver elé az IP láncban? Ahol az IIS lenne a frontend. Az IIS-nek ebben az esetben egyrészt át kell fordítania és továbbítani a kérések egy részét a VISION web szervere felé, másrészt ellátni egyfajta fordított proxy szerepet. Bonyolultnak hangzik, de a megoldás egyszerű:

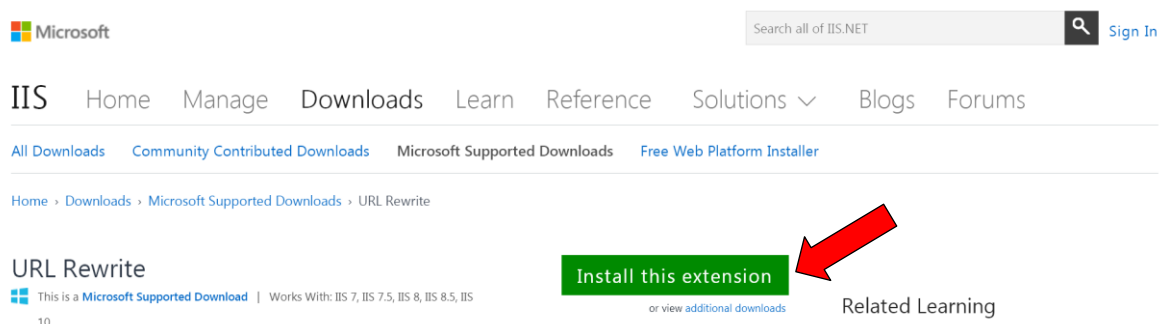
1. az IIS oldalon be kell konfigurálni az **URL rewrite** szolgáltatást
2. a VISION-t át kell kapcsolni relatív címzésű üzemmódba (új kapcsoló)

Az utóbbi az egyszerűbb: egy új checkbox:



Az URL rewrite azonban bonyolultabb kisé. Mindenek előtt a funkciót kell installálni, ha még nem létezne:

<https://www.iis.net/downloads/microsoft/url-rewrite>



Majd feltenni még egy komponenst: az *Application Request Routing* szolgáltatást is (reverse proxy):

Microsoft

Search all of IIS.NET Sign In

IIS Home Manage Downloads Learn Reference Solutions Blogs Forums

All Downloads Community Contributed Downloads Microsoft Supported Downloads Free Web Platform Installer

Microsoft Supported & Community Downloads for IIS...

Microsoft downloads are fully supported with future updates, bug fixes and customer support. Community downloads are submitted by IIS community members and do not benefit from Microsoft approval or support, and should be downloaded with this in mind.

Get the Web Platform Installer

Most Microsoft Downloads can be installed using Web Platform Installer however it is not required.

A szolgáltatás a *Web Platform Installer*-en keresztül érhető el. De internet eléréssel nem rendelkező szerver környezetben offline is feltelepíthetjük. Az offline installer innen tölthető le:

<https://www.microsoft.com/en-us/download/details.aspx?id=47333>

Telepítsük fel mindkét szolgáltatást! A beállítás ezek után a következő:

1. lépés: nyissuk meg az *Application Request Routing Cache* szolgáltatást az IIS-en belül:

The screenshot shows the IIS Manager console with the 'IIS' area selected. The 'Application Request Routing Cache' feature is highlighted with a red arrow. The 'Connections' pane on the left shows the hierarchy: SERVER (Server:InternetServer) > Application Pools > Sites > Default Web Site > IIS.

2. lépés: Server Proxy Settings

The screenshot shows the 'Application Request Routing Cache' feature selected. The 'Server Proxy Settings' feature is highlighted with a red arrow. The 'Connections' pane on the left shows the hierarchy: SERVER (Server:InternetServer) > Application Pools > Sites > Default Web Site > IIS > Application Request Routing Cache.

3. lépés: Enable proxy (a sárga háttérű szövegekkel ne törődjünk!)

The screenshot shows the 'Server Proxy Settings' feature selected. The 'Enable proxy' checkbox is checked and highlighted with a red arrow. The 'Connections' pane on the left shows the hierarchy: SERVER (Server:InternetServer) > Application Pools > Sites > Default Web Site > IIS > Application Request Routing Cache > Server Proxy Settings.

4. lépés: Nyissuk meg az URL rewrite szolgáltatást:

The screenshot shows the 'Default Web Site Home' feature selected. The 'URL Rewrite' feature is highlighted with a red arrow. The 'Connections' pane on the left shows the hierarchy: SERVER (Server:InternetServer) > Application Pools > Sites > Default Web Site > IIS > URL Rewrite.

5. lépés: Adjuk hozzá ezt az *Inbound* szerepet (a lényeg itt az **alkalmazás neve**, meg az **IP:port**):

The screenshot shows the 'Match URL' dialog box. The 'Requested URL' is 'Matches the Pattern'. The 'Using' dropdown is 'Regular Expressions'. The 'Pattern' field is 'AlmalmazasNeve/{.*}'. The 'Rewrite URL' field is 'http://192.168.1.100:8080/{R:0}'. The 'Append query string' checkbox is checked. The 'Stop processing of subsequent rules' checkbox is checked.

Már csak egy IIS újraindítást kell végrehajtanunk és készen is vagyunk. Azt értük el, hogy az IIS-be befutó kérések közül azokat, amelyek `/AlkalmazasNev/...` hivatkozást tartalmaznak, automatikusan átirányítja az IIS a Rewrite URL-nél megadott IP- és Port címre. Ami pedig pont egy VISION web alkalmazásnak is a követelménye, hisz a VISION-ben az összes hivatkozás a projekt nevével kezdődik. Abban az esetben, ha több projektünk is van, akkor mindet fel kell venni az URL rewrite Inbound szerepek közé.

Nem olyan veszedelmesen bonyolult, de legalább itt pontról pontra össze van foglalva, nem úgy, mint az interneten! Még a teljesség kedvéért megmutatom a Web.config beállítását is, hisz kézzel beírva gyorsabban elérhetjük a kívánt eredményt. Ez szerintem sokkal egyszerűbb is, kevesebb hibalehetőséggel:

```
<rewrite>
<rules>
<rule name="VISION Proxy Inbound" stopProcessing="true">
  <match url="AlkalmazasNev/(.*)" />
  <conditions logicalGrouping="MatchAll" trackAllCaptures="false" />
  <action type="Rewrite" url="http://192.168.1.100:8080/{R:0}" appendQueryString="true" logRewrittenUrl="false" />
</rule>
</rules>
</rewrite>
```

A web.config az `c:\inetpub\wwwroot` alatt található. Átírás után az IIS-t újra kell indítani.

Windows 10 alatt azért van egy kis probléma. Mégpedig az, hogy az URL rewrite-ot nem lehet rá feltelepíteni, mert a Microsoft nem figyelt eléggé oda. A megoldás, hogy telepítés előtt a fő verziót 9-re kell állítani, majd installálás után vissza 10-re. Durva, valaki napokat töltött el, hogy ezt kiderítse ☹

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\InetStp\MajorVersion = 9, majd 10
```

A teljesség kedvéért megemlítem, hogy az URL rewrite fenti beállítása még nem teljes, mivel csak a már említett `/AlkalmazasNev/...` típusú URL-eket irányítja át, de pl. ezt nem: `/AlkalmazasNev`. Ami ugye az alaphivatkozás, szóvégi slash / nélkül! Nem lenne baj, ha menne ez is.

Küzdelmes próbálkozások tömege után a következő egyszerű, trükkös megoldás született: **fel kell venni az AlkalmazasNev-et a virtuális könyvtárak közé is**, de az már mindegy, hogy a könyvtárban mi van. Ilyenkor az IIS önmagát veri át, hisz a default / hivatkozást automatikusan hozzáfűzi az URL végéhez, onnan pedig már az URL rewrite irányít. Mivel azonban az alkalmazás nevét mindenképp meg kell adni a hivatkozásban, egyszerűbb, ha eleve a főképre irányítjuk a programot, kikerülve a VISION saját kezdőképét.

Végül ne felejtjük el törölni az előzménytárat, amikor a két üzemmód között váltogatunk, mivel a tallózónak a két esetben teljesen más tartalmú javascripteket kell cache-elnie.

Megmértük, mennyit lassít az IIS a rendszeren: requestenként 3-4 ms-ot.

Az egyik legfontosabb előny, amit a fenti beállításokkal elérhetünk, hogy a program innen kezdve SSL/TLS titkosítással is működőképes, vagyis **a https átirányítás is automatikusan megoldott**. Nem kell tehát PFX fájlok exportálásával és a visHTTPS alá történő beimportálásával bajlódni. Ami nagyvállalati környezetben egyébként se nagyon preferálnak (hisz tanúsítványokat nem illik senkinek sem odaadni, jelszót megadni). De fontos előny az is, hogy a vállalati rendszerbe integráltuk az alkalmazást, egy közös frontend kiszolgáló mögé, ami önmagában rendelkezik már az összes szükséges védelemmel és felügyelettel - a HTTPS szolgáltatáson túl (IPsec, loadbalancer, vírus- és túlterhelés védelem, request logging...). Az egész folyamatot *en bloc* a rendszergazda felügyelheti, aki egyébként is felel a tanúsítványok telepítéséért és megújításáért. Nekünk ezzel innen kezdve nincs semmi dolgunk.

4. Dropbox használata

Egyre több web-es alkalmazásban igény, hogy fel lehessen tölteni fájlokat a szerverre, ami aztán feldolgozza azokat. Például dokumentumokat, csv vagy xls(x) fájlokat, képeket lehet így feltölteni.

A megoldás viszonylag egyszerű. Magában a VISION alkalmazásban, a képen egy IB (Internet Browser) objektumot kell elhelyezni, az URL-be pedig beírni, hogy *Dropzone/Dropbox.html*:

Azonosítás	
Név	IB1
Csoport	GRAPH
Szülo	
Lapazonosító	*
Megjelenés	
Pozíció	43,31
Méret	65,25,22,125
URL	Dropzone/DropBox.html
Töröl ha láthatatlan	False
Láthatóság	True
Megj. vezérlése	

A képen ez jelenik meg (Web szerver aktív kell hogy legyen):

Kattintson a területre az fájl feltöltéséhez, vagy dobja ide (drag-&-drop) a fájlt.

Persze csak azután, hogy az alkalmazásba a VISION telepítési helye alatt a Web\Root\Dropzone könyvtárat ugyanilyen néven (Dropzone) bemásoltuk. Ez azért nem automatikus, mivel a DropBox.html fájlt gyakran kézzel módosítjuk az alkalmazás igényei szerint. Például a fenti *"Kattintson a területre..."* szöveg miatt.

A dropbox-ba bedobott fájlokat a VISION egy Fifo-ban tárolja el, abból olvashatja ki az alkalmazói program. E célra négy függvény született:

```
LastUploadedFile:: namestring function 'csak kiolvas
GetUploadedFile:: namestring function 'kiolvas és töröl
ClearLastUploaded:: procedure 'töröl
ReadUploadedFile:: procedure (filename: var string; username: var namestring;
                             password: var namestring; SID: var long; cancel: var ushort)
```

Azt, hogy egyáltalán van-e bármi a Fifo-ban, a legegyszerűbben a *LastUploadedFile* függvény tesztelésével végezhetjük el. Ez mindig visszaadja a Fifo legelső elemét (a Last itt megtévesztő lehet, de ez Fifo). Viszont nem veszi ki onnan a fájlt (nem lépteti a Fifo-t), csak a *GetUploadedFile*, ami kiolvassa az elemet és tovább is lép. Külön törölni a *ClearLastUploaded* függvénnyel is lehet. Valamint a részletes információt szolgáltató *ReadUploadedFile* függvénnyel. Ez utóbbi nem csak a fájl nevét adja vissza (filename), de a felhasználó nevét, jelszavát és Session azonosítóját is, amennyiben a felhasználó web-en lép be és használja a Dropbox-unkat. Viszont ez utóbbi további, speciális beállításokat is igényel a web-es felülettől, amire itt nem térnek ki.

A következő példa megmutatja, hogyan lehet a legegyszerűbben használni:

```
Refresh      if LastUploadedFile<>" " then
                LastUpload=ProgDir+"Tmp\Upload\"+LastUploadedFile;
                Kep=GetUploadedFile;
                CopyFile (LastUpload, Pictures+"\"+Kep);
            fi
```

Ciklikusan azt teszteljük, hogy van-e a Fifo-ban egyáltalán bármi (*LastUploadedFile<>" "*), s ha van, akkor abból előállítjuk a forrásfájl teljes nevét. A fájlokat a program a saját *Tmp* könyvtára alatt egy *Upload* alkönyvtárban őrzi, ezért kell a fenti *LastUpload=* értékadás. Majd a fájlt magát egy *Pictures* alkönyvtárba bemásoljuk. Maga a Fifo a *Kep=GetUploadedFile* paranccsal törlődik és jól mutatja, hogy miért van rá szükség. Ha megfigyeljük, a *LastUploadedFile*-ra kétszer is hivatkozunk, mielőtt azt a Fifo-ból törölnénk. Csak a legutolsó hivatkozás, a *Kep=Get...* értékadás törli, ergo a *LastUploadedFile* függvény mostantól üres string és az IF szakasz sem aktív többé. Várjuk a következő képet...

5. ChartTrend SQL adatbázis web-es megjelenítése (ChartID)

Mint ismert, a TD és TDX objektumok SQL adatbázis táblák közvetlen megjelenítésére is alkalmasak, úgy desktop-on, mint web-en. Azonban ez utóbbi esetben probléma lehet, hogy a hozzárendelt változó, ill. tábla nem változik (hiszen vagy nincs vagy mindig ugyanaz), ezért csak desktop-on változik a trendkép, ChartTrend-et bekapcsolva, azaz web-en már nem feltétlenül. A jelenségnek egyszerűen az a magyarázata, hogy az objektum megjelenítésére előállított HTML fájl neve nem változik, mivel az csak a kezdő és végidőt, az objektum-megjelenítés vezérlő tulajdonságait és a változónévből (táblanévből) generált kódot tartalmazza, így:

VT-653011200 – 653051440 – 8033 - 724572.htm
kezdőidő végidő mód változó(tábla)

Az SQL tábla-hivatkozás tehát változatlan fájlnevet eredményez változó tartalom mellett. A tallózó így az előzőleg cacheelt tartalommal fog dolgozni és a kép nem változik.

Hogy rövidre fogjam, a megoldás a következő: bevezetésre került egy új TD/TDX tulajdonság: **ChartID**



Ez egy long típusú mező (32-bit integer), ahová egy kódot kell generálnunk, mely a mérést egyedileg azonosítja. A példában egy olyan változó került beírásra, mely ebben az esetben a mérés sorszáma.

A ChartTrend tábla neve ezzel az azonosítóval bővül (meres_srsz=6):

VT-653011200 - 653051440 – 8033 – 724572 - **6**.htm

Ezáltal viszont web-en is változni fog a tartalom. Véletlenszám beírása nem javasolt!

A témához kapcsolódóan van még egy extra opció, amivel azt érhetjük el, hogy a session azonosító is bekerüljön a fájlnevébe. Egy új VPR.INI bejegyzés:

SessionWebChart=1

Azt azért fontoljuk meg, hogy ezt tényleg akarjuk-e. Hisz mostantól minden egyes login-nak önálló verziója lesz (miközben a tartalom a session azonosítót leszámítva több fájlban is azonos).

6. SQL trend szinkronizáció

Azoknál az alkalmazásoknál, ahol az egymással hálózati (DFS) kapcsolatban álló munkaállomások mindegyike SQL trend kezelést használ, felmerülhet az igény, hogy az adatokat a távoli állomásról átmásoljuk, szinkronizáljuk – mert elképzelhető, hogy a távoli szerveren frissebb az adat. Valójában egyszerű táblaműveletekre van szükségünk, amely akár a *SyncDatabase* paranccsal is megoldható lenne. A baj csak az, hogy a trend esetében nagyszámú tábláról van szó és igazából nem is tudjuk, hogy mennyiről. A probléma megoldására egy könnyen használható eljárás született:

```
SQLTrendRead:: procedure (remoteUDL: namestring; timefrom: long; timeto: long)
```

A parancs mindössze a távoli trend SQL szerverének UDL fájlját igényli és persze az adatbázist azzal el kell tudni érni. Ezen kívül meg kell adni az idő kezdetét és végét. A legegyszerűbb úgy használni, hogy készítünk egy szervíz képet, ahol beviteli mezővel határozzuk meg az idő kezdetét és végét. Majd manuálisan elindítjuk a műveletet.

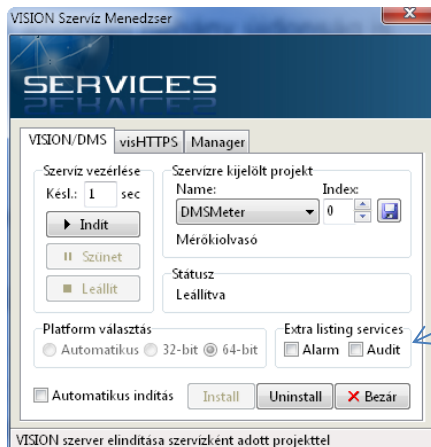
X10 Verzió 2.23

1. Módosítások szervízekben

A szervízkezelés tekintetében számos új fejlemény van. A VSM programot mostantól *SetupServices*-nek hívják. Feltelepítés után a program menü is így hivatkozik rá. Ez egyrészt beszédesebb, másrészt emelt szintet (administrator) forszíroz, ami szükséges feltétele a beállításoknak. A működése nem változott: processzként fut és a tálcán az ikonsorban jelzi egy ikon:



Erre kattintva lehet megjeleníteni, egy újabb kattintással pedig elrejteni a beállító felületet:



A beállítás alapvetően nem változott, továbbra is az *Index* léptetésével lehet hozzárendelni az egyes szervízpéldányokhoz a projektet, maximum 10-et. Nincs ugyanakkor automatikus mentés – ezt a felhasználók kérték! – csak a mentés gomb megnyomásával lehet lementeni a beállításokat, külön az egyes indexekhez. Van azonban néhány újdonság is.

Extra listing services:

- Alarm
- Audit

A VSM.ini fájlba a rendszer ezeket a beállításokat *AlarmList* és *AuditTrail* néven menti el. Szerepe pedig az, hogy amennyiben nem kérjük, ezeket a szolgáltatásokat a szervíz el sem indítja.

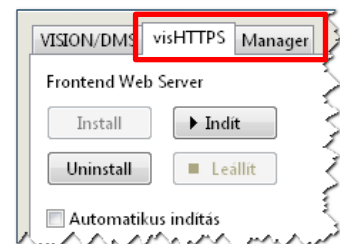
Ez az opció az alarm szervízt kikapcsolja, lévén az alarm listázónak része. Nélküle nem fut az alarm taszk sem. Ami a listázó funkciót illeti, szervíz szinten megjeleníteni sem az alarm listát, sem az Audit Trail-t nem lehet, ezért emiatt önmagában nem szükséges a szolgáltatás, legalábbis a kliens szervízekben, de a server szervízben szükséges, mert különben nem keletkeznek alarmok. A kliens szintű alarm funkciók (nyugtázás, bejelentkeztetés stb.) azonban működnek nélküle is.

A másik fontos különbség, hogy megjelent két új fül és vele két új szervíz:

- visHTTPS
- Manager

Az előbbi azt jelenti, hogy a VISION *Frontend web* szerverét ezentúl a szervíz menedzserből is elindíthatjuk (eddig csak a *_visHTTPS.exe* applikációval lehetett). Egy helyen állítható tehát az összes szervíz.

A *Manager* szolgáltatás, pontosabban Manager szervíz pedig egy apró, ugyanakkor nagyon hasznos eszköz a szervízek közös kezelésére és működésüknek a programból történő ellenőrzésére, sőt vezérlésére.



1.1. Manager szervíz

Az új szolgáltatás fő feladata a szervízek ellenőrzése, egyfajta *guard* szerep, valamint az automatikus indítási, újraindítási és leállítási folyamatok vezérlése. Ezt a programocskát a program telepítéskor automatikusan installálja és el is indítja, ill. gépjúraindítás után is mindig futni fog. Leállítani, esetleg uninstallálni csak a szervíz menedzserben lehet, a Manager fülön. A következő feladatokat látja el:

- Automatikus upgrade folyamat menedzselése
- Képes újraindítani a szervízeket, amennyiben azok leállnának (*guard* szerep)
- Képes az összes szervíz elindítására, amennyiben azok nem automatikus (manuális) módban lettek feltéve, de később automatikus indítás beállítást eszközöltünk akár a szervíz menedzserrel, akár a VSM.ini program sorainak kézi, vagy alkalmazói programból való

átírásával (AutoStart=1). Más szóval a Windows helyett rábízhatjuk az automatikus indítást a menedzser szervízre is.

- A menedzser szervízen keresztül lekérdezhető a szervizek állapota VISION alkalmazói programból, sőt vezérelhető: áll, indít, újraindít, uninstall, install.

Felvetődik a kérdés, hogy mi szükség van a szervizek *Manager*-ből történő indítására a Windows helyett? A válasz az, hogy ez utóbbihoz **nem szükségesek adminisztrációs jogok** – másképp az alacsony szinten futtatott VISION alkalmazásból még a státuszához sem férnénk hozzá! Másrészt a Manager szervíz biztonságosabb is, hisz ellát ellenőrző szerepet és támogatja az automatikus update-et, továbbá hozzáférést enged a VISION alkalmazás számára. Nézzük részleteiben.

Az alábbi képernyőmentés egy feltelepítés utáni tipikus állapotot mutat a *Windows Services* listájában. A lista 3db szervízként üzemeltetett VISION projektet mutat és tartalmazza a visHTTPS frontend szolgáltatást is, ami a web alkalmazásokhoz szükséges. A felsorolás legvégén látható a *Vision Service Manager*, ami el is van indítva.

Name	Description	Status	Startup Type
Vision HTTPS Service	Vision's HTTP(S) Frontend Web Server.		Manual
Vision Server	Vision Application Server running as a service.		Manual
Vision Server 1	Vision Application Server running as a service. Additional service instance #1		Manual
Vision Server 2	Vision Application Server running as a service. Additional service instance #2		Manual
Vision Service Manager	Vision Service Manager	Started	Automatic

Mint látható, az összes szervíz *Manual* módban van, csak a *Vision Service Manager* automatikus. **Ez a javasolt beállítás!** Ezek után, ha a VSM.ini fájlban az AutoStart bejegyzést 1-re állítjuk, a menedzser el fogja indítani a szervízt gépjáraindítás után. Egyébként nem. Nem kell tehát többé hozzáférnünk a szervizek listájához és nem kell adminisztrációs jog sem a művelet elvégzéséhez.

Már most hogyan tudjuk mindezt VISION alkalmazásból állítani, monitorozni? Készült erre egy példa, ami az AdvancedCourse alkalmazásban megtekinthető:

Szervíz menedzser

Szolgáltatások	Státusz	Parancsok
Frontend:	Fut...	Indít Leáll
Mérőkiolvasó:	Fut...	Indít Leáll
Tartalomszolgáltató:	Fut...	Indít Leáll
Riport, dashboard:	Leállítva	Indít Leáll

Működése négy új VISION parancsra épül:

- *GetServiceStatus*
- *GetServiceInfo*
- *CommandService*
- *TriggerService*

Leírásuk megtalálható a *Programming Reference Manual*-ban. Alább egy képernyő mentés látható a függvény prototípusokról:

```

GetServiceStatus::    ushort function (serviceorproject: shortstring)
GetServiceInfo::      string function (serviceorproject: shortstring; aboutwhat: shortstring)
CommandService::      procedure (serviceorproject: shortstring; command: shortstring)
TriggerService::       procedure (serviceorproject: shortstring; index: integer; timeout: long)

```

Például a **Fut...** állapotot a *GetServiceStatus=2* kód jelzi. Ha nem fut a manager, minden státusz 0. Paraméterként egyaránt átadhatjuk a szervíz nevét vagy a hozzárendelt projekt nevét (*serviceorproject*), ami nagyon kényelmessé teszi a használatát. A *CommandService*-en keresztül pedig kiadhatunk egy csomó parancsot: *install*, *uninstall*, *start*, *stop*, *restart* – így, szövegesen. Maguk a függvények rendkívül gyorsak, ezért ciklikusan a képből is hivatkozhatunk rájuk.

Mikor lehet szükségünk arra, hogy a háttérben futó szervízt újraindítsuk programból? Gondoljunk arra, hogy módosítjuk a változókat egy VISION kliens alkalmazásban, vagy vettünk fel új ciklikus programokat, kommunikációt, alarmot; esetleg bemásoltuk kézzel az új VAL fájlokat a szerver projekt könyvtárába. A szervízt ilyenkor újra kell indítani, hogy a módosítások érvényre jussanak! Ezért célszerű tehát egy a fentihez hasonló kis template-et valahol magában az alkalmazásban elhelyezni. Mindezt megcsinálhatjuk úgy, hogy ki se kelljen lépni a kliens alkalmazásból...

1.2. Guard szolgáltatás használata

Mint a bevezetőben említettem, a *Vision Service Manager*-t a Vision szervizek biztonságos működtetésére is fel lehet használni. Nem csak vezérelni lehet tehát a szervízt közvetlen parancsokkal (áll, indít, install, stb.), de letölthetünk egy vagy több életjel triggert is a menedzserbe magából az alkalmazásból.

```
TriggerService(serviceofproject,index,timeout);
```

ahol

serviceorproject: A szervíz neve (*VisionServer*, *VisionServer1*, *VisionServer2*...), vagy a projektünk konkrét neve (pl. *KecskemetiVizmu*)

index: trigger indexe 0..9 (egy-egy szervíz legfeljebb 10 dolgot figyelhet)

timeout: másodperces érték annak meghatározására, hogy a legközelebbi *TriggerService()* hívásnak mennyi időn belül kell újból bekövetkeznie

Képzeld el, hogy a szervíz menedzserrel azt ellenőrzítjük, hogy a kommunikáció sikeres-e egy percen belül. Ezt vizsgálандó a sikeres beolvasások eseménykiszolgálójába beírjuk, hogy

```
TriggerService("KSZVTT",0,60).
```

A parancs hatására létrejön egy időzítő a szervice menedzserben az adott projekthez rendelve a 0-ás indexen. A beállított idő = 60 másodperc. Ezután a szervíz menedzser elkezd visszaszámlálni másodpercenként. Amennyiben nem kerül sor újabb *TriggerService()* hívásra 1 percen belül, úgy a szervíz menedzser a szervízt újraindítja.

Figyelhetjük a triggerrel a ciklikus programunkat, a vezérlőt, a kommunikációt, az alarmot, az adatbázisok elérhetőségét – lényegében minden olyan taszkot, aminek a szervízben biztosan futnia kell.

Egyébként azért van szükség alkalmazás szintű megoldásra, mivel általában azt nehéz megállapítani, hogy a szervíz működése mikor egészséges. Hisz vannak olyan alkalmazások is, ahol a VISION csak naponta egyszer csinál valamit, és az sem biztos, hogy a kommunikációnak mindig működnie kell, lévén ki is kapcsolhatnak bizonyos alrendszereket. Nem lenne szerencsés ilyenkor szervízt újraindítgatni automatikusan. Ezért csak alkalmazás szinten lehet azt eldönteni, hogy mit és mikor kell figyelni – ha bármit egyáltalán!

Végül van még egy beállítás, amivel a szervíz működési biztonságát növelhetjük. Egy további paraméter a VSM.INI-ben:

```
MustRun=1
```

Amennyiben az INI fájl ezt tartalmazza, a szervíz menedzser nem csak induláskor indítja el a szervízt (*AutoStart=1*), de minden olyan esetben, ha a szervíz bármilyen okból megáll. Ezt megint nekünk kell azonban eldönteni, hisz a szervízt akár kézzel is leállíthatjuk – például rendszerfrissítés miatt. Elég durva lenne, ha ezt nem tudnánk elkövetni, mert a szervízmenedzser azzal bosszantana bennünket, hogy a szervízt folyton visszaindítja.

Egyébként, ha kézzel vagy programból módosíthatjuk a VSM.INI fájlt – a szervízmenedzser beolvassa automatikusan. Azt tehát nem kell emiatt újraindítani. Sőt soha nem kell újraindítani. Menet közben is bármikor módosíthatjuk akár a *MustRun* paramétert, akár az *AutoStart*-ot, a szervízmenedzser kimentés után szinte azonnal be is olvassa a módosított fájlt, majd értelmezi annak tartalmát. Mi több, ezekről a trace.log fájlban bejegyzést is készít. Meg is mutatom a következő pontban.

1.3. Szervízek használata, tapasztalatok

Az utóbbi pár évben összegyűlt némi tapasztalat a szervíz üzemmódban működtetett VISION rendszerekkel kapcsolatban. Nézzük meg azt először, hogyan lehet ellenőrizni, hogy minden rendben betöltődött-e, majd vegyük sorra, hogy mire kell ügyelni.

1.3.1. Log

A program telepítési könyvtára alatt a Bin\Tmp\ben található egy **trace.log** nevű log file. Ennek tartalma egy egészséges szervíz betöltési és indítási folyamat végén a következő:

```
Service: Index: 0, Name: VisionService, DisplayName: Vision Server, ExeName: VisionServer
Service: detected DPI = 96, hres = 1024, vres = 768
Service: End of main initialization
E:\VISIONX\Bin\
> START TestProject
(0) Start Audit Trail
(1) Start loading project
(2) Begin building of project
(3) Creating web server form...
(4) End building of project
(5) Finish project tree
(6) Valid application: 1
(7) Building project ok
(8) Setup networking
(9) End setup networking
(10) End loading project (Prjnum=2)
(11) Initializing project
(12) Web server enabled
(13) Initialize both alarms and Audit Trail
(14) Creating alarm listing form...
    Last alarm: 2021.02.14 13:13:14
(15) Over initializing alarms (Audit Trail)
(16) End of reading trend, acquisition, alarms, users
(17) Start networking
(18) End start of networking
> STOP
    Stop communication
    Shutdown communication
    Shutdown project
> DESTROY
```

A betöltési folyamat egyes fázisait sorszámozzuk az új rendszerben, hogy könnyebb legyen megállapítani, vajon sorrendben helyesen indultak-e el a folyamatok és lefutott-e mind a (18)-as pontig.

Természetesen előfordulhat, hogy a (0) és a (13)-(15) kimarad, lévén ezeket az új beállításokkal (Audit Trail és Alarm listázás) kikapcsolhatjuk, és persze a (12) is elmaradhat, ha nincs Web szerver. A fenti a full betöltési lista. Végül több projekt esetén a (11)-(16) ismétlődik.

A lista alján az egészséges leállítási folyamat látható.

1.3.2. SQL adatbázisok kezelése

Az OLE-DB típusú SQL adatbázis kapcsolatok esetén arra kell vigyázni (tipikus hiba!), hogy szervízben a Windows autentikáció nem használható. Csak felhasználó névvel és jelszóval lehet az MS-SQL szolgáltatásait igénybe venni! Különben a betöltési folyamat meg is áll.

Érdekes módon a közvetlen elérésű (DAC) adatbázisok azonban működnek Windows autentikáció mellett is. De nem csak ezért érdemes őket használni inkább. Tudni kell, hogy az OLE-DB valójában egy ActiveX szolgáltatás, aminek a használata problémás lehet szervíz üzemmódban. A program ugyan mindent megtesz azért, hogy ez több szálon is képes legyen futni, és fut is az esetek 99%-ban, de sajnos nem mindig (ld. 1.2.3.). Viszont a közvetlen elérésű adatbázisok mindig működnek!

1.3.3. Tapasztalatok különféle op. rendszerekkel

A VISION szervízek minden Windows operációs rendszeren működnek, de nem egyöntetűek a tapasztalatok. Ezeket foglalja össze a következő táblázat:

Op. rendszer	Tapasztalat	Megoldás
Microsoft Windows 7	Minden funkció tökéletesen fut és a rendszer kellően gyors is.	-
Microsoft Windows 10	Minden funkció tökéletesen fut, de a szervízek listázása jóval lassabb, mint Windows 7-ben	Lehet vele élni
Microsoft Windows Server 2003	Minden funkció tökéletesen fut és a rendszer kellően gyors is.	-
Microsoft Windows Server 2016+	A rendszer ugyanannak a szervíz példánynak a 2. 3. stb. változatát <u>nem mindig</u> tölti be, az OLE alapú szerver szolgáltatásokra jellemzően csak egy példányban lehet hivatkozni: a szerver alkalmazásban.	Az Alarm és Audit Trail listázó funkciót, ami a szerver taszk, ki KELL kapcsolni (többek között ezért készült!)

1.4. Szervízekkel kapcsolatos változtatások a VISION projektben

A szerver – kliens rendszerek vegyes kezelését segíti egy új konfigurációs opció:

Egy új automatikus szerver-kliens váltókapcsoló került beépítésre, ami szó szerint azt csinálja, ami bele van írva: **mindenképp szerverként indítja az alkalmazást, ha szervízként fut és mindenképp kliensként, ha a szervíz automatikus indításra installált.** Így ugyanazt a VPR változatot használhatjuk mindkét célra: szerver ha szervíz, kliens, ha nem szervíz. Vagyis ugyanazzal a konfigurációval indíthatunk rá klienst. Ám nem csak akkor, ha már fut (mint a normál automatikus hálózathál), de akkor is, ha "csak" installált és az AutoStart=1 a VSM.ini fájlban. Lévé a szervíz elindulhat később is, mint a kliens – így tehát biztonságosabb a működtetés. Az új opció csak az előtte látható rádiógommbal együtt határos: *automatikus hálózat*.

X10 Verzió 2.22

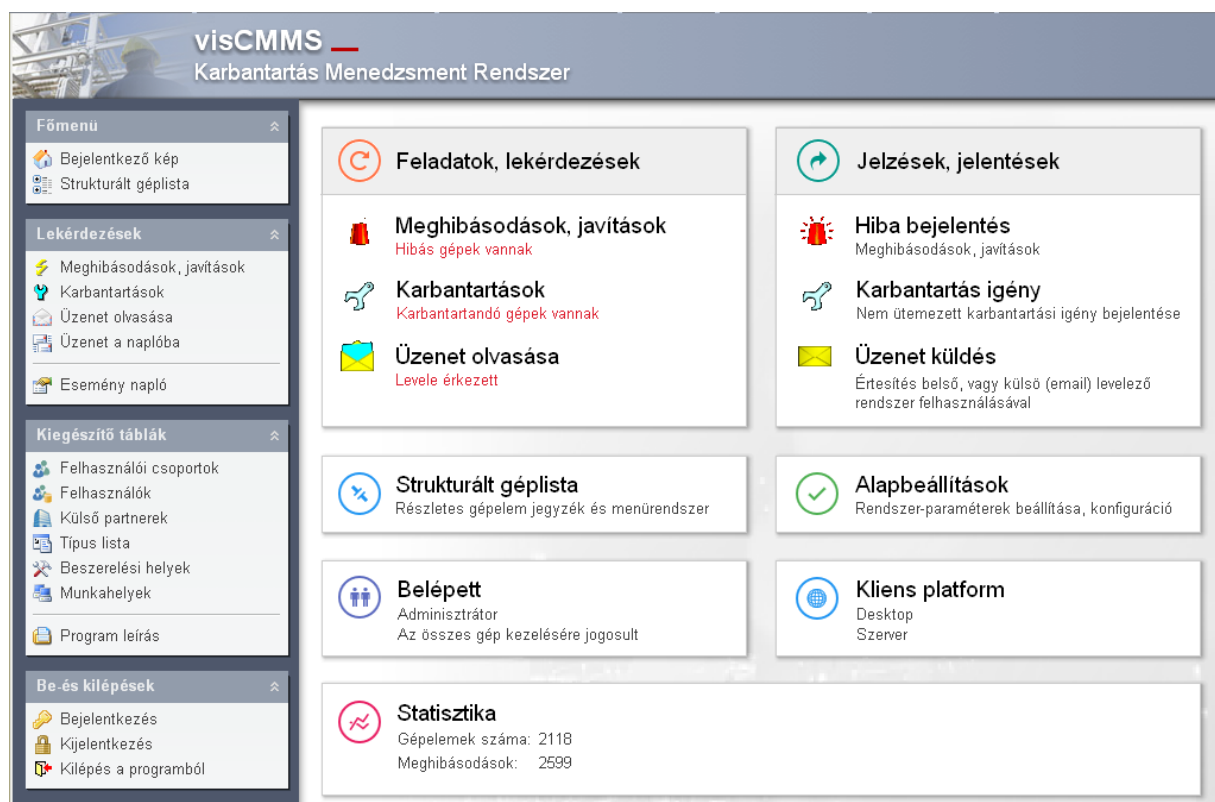
1. VISION X10 alapú karbantartás menedzsment

Komoly fejleménynek és előrelépésnek számít a teljesen VISION alapú karbantartás menedzsment rendszer kifejlesztése. A következők miatt:

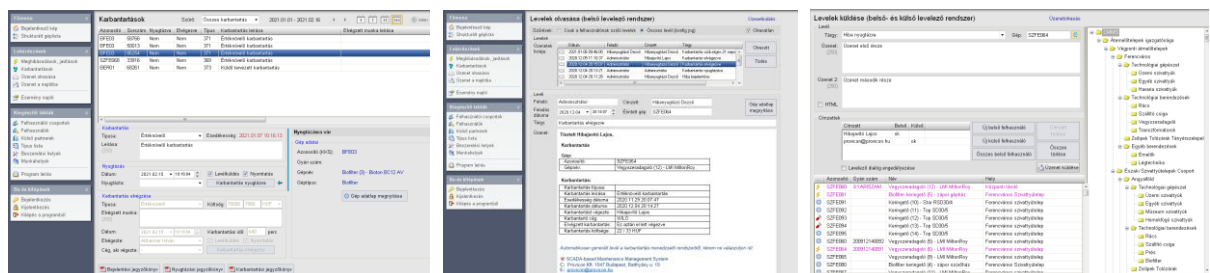
- Az új CMMS teljesen web-es
- Minden korábbinál egyszerűbb meglévő VISION alkalmazásokhoz integrálni

Ez utóbbi kitétel egyszerűen azt jelenti, hogy összevont alkalmazásként lehet hivatkozni rá. A ServiceWindow új változata pedig már ezekre az adatokra épít. Az alkalmazásban a hibaállapot megjelenítés, hibák, karbantartások indítása és a munkalapok megtekintés soha nem volt még ilyen egyszerű.

Az eddigieken túl az új program sokkal látványosabb és gyorsabb is (a VISION X10 miatt), mint elődje.



Felépítése és a működése ugyanazokra az alapokra épít és az egyes formok is nagyon hasonlóak. A rengeteg különálló ablakot azonban template-be ágyazott technika váltotta fel, ami a web-en is sokkal használhatóbb:



Az esemény lista már teljesen a VISION saját esemény listája:

The screenshot displays the visCMMS Karbantartás Menedzsmenet Rendszer interface. The top bar shows the date 2020.11.01 - 2020.12.01 and the user Adminisztrátor. The left sidebar contains navigation options like Főmenü, Lekérdezések, Kiegészítő táblák, and Be- és kilépések. The main area shows an event log for the date 2020.11.15, 14:05:22. The log table has columns: Tp, Datum/Idő, Tag, Üzenet, Érték, and Típus. The events are categorized by Tag (visCMMS) and Típus (Befejezés, Indítás, Hiba, Rendszer, Technológia).

Tp	Datum/Idő	Tag	Üzenet	Érték	Típus
2020.11.15. 13:43:15	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 13:43:20	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 13:43:55	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 13:44:01	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 13:44:50	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 13:44:57	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 13:46:01	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 13:56:26	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 13:58:14	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 14:01:16	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 14:03:54	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 14:04:04	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 14:05:22	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 14:05:28	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 14:06:04	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 14:06:10	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 14:07:51	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 14:07:57	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 14:08:23	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 14:08:29	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 14:09:47	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 15:34:40	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 16:01:33	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 16:01:48	visCMMS	Projekt indítása	visCMMS	Indítás	
2020.11.15. 16:02:37	visCMMS	Projekt befejezése	visCMMS	Befejezés	
2020.11.15. 16:03:32	visCMMS	Projekt indítása	visCMMS	Indítás	

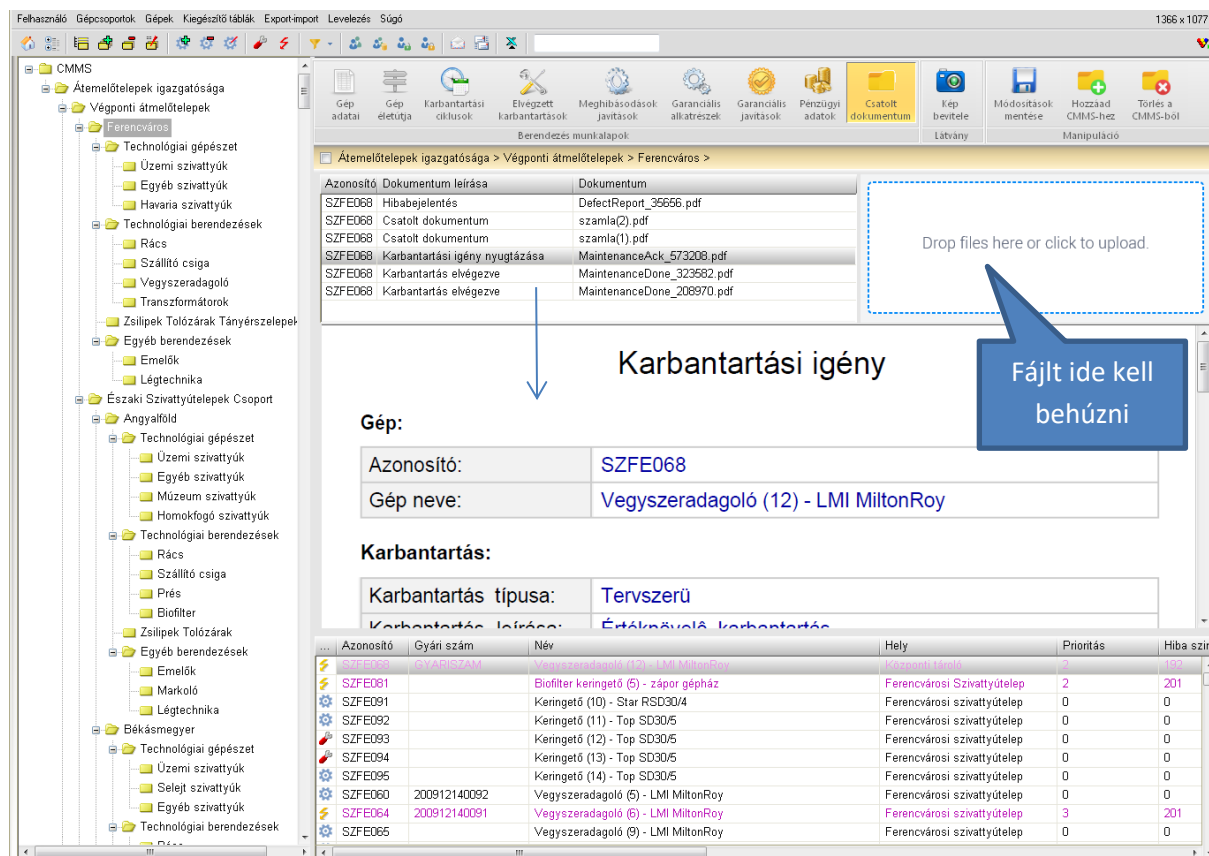
A leglátványosabb változás azonban a beépült gépjegyzék kezelés:

The screenshot displays the VISION X10 interface for equipment management. The left sidebar shows a tree view of equipment categories. The main area shows the details for a specific piece of equipment, including its identification, parameters, and maintenance status. A large image of the equipment is shown on the right. Below the details, there is a table listing all equipment in the system.

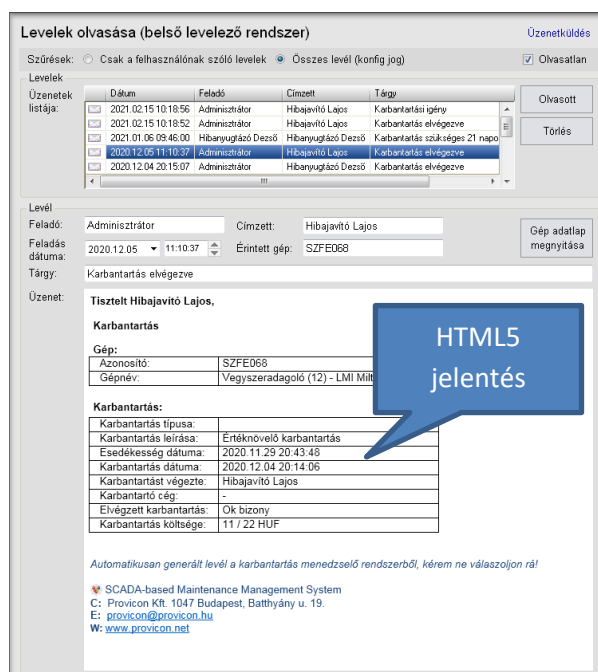
Azonosító	Gyári szám	Név	Hely	Prioritás	Hiba szint
SZFE08	GYARISZAM	Vegyszeradagoló (12) - LMI MiltonRoy	Központi tároló	2	192
SZFE081		Biofilter keringető (5) - zápor gépház	Ferencvárosi Szivattyútelep	2	201
SZFE091		Keringető (10) - Star RSD30/4	Ferencvárosi szivattyútelep	0	0
SZFE092		Keringető (11) - Top SD30/5	Ferencvárosi szivattyútelep	0	0
SZFE093		Keringető (12) - Top SD30/5	Ferencvárosi szivattyútelep	0	0
SZFE094		Keringető (13) - Top SD30/5	Ferencvárosi szivattyútelep	0	0
SZFE095		Keringető (14) - Top SD30/5	Ferencvárosi szivattyútelep	0	0
SZFE060	200912140092	Vegyszeradagoló (5) - LMI MiltonRoy	Ferencvárosi szivattyútelep	0	0
SZFE064	200912140091	Vegyszeradagoló (6) - LMI MiltonRoy	Ferencvárosi szivattyútelep	3	201
SZFE065		Vegyszeradagoló (9) - LMI MiltonRoy	Ferencvárosi szivattyútelep	0	0

Itt minden egy képen állítható. Kiválaszthatunk másik gépcsoportot, abból az alsó listából gépet és állíthatjuk az összes paramétert (adatlap, életút, hiba stb.) az office stílusú menü elemeire kattintva.

Mivel pedig a program web-es, a fotókat és csatolt dokumentumokat egyszerűen drag-&-drop művelettel helyezhetjük el a gép munkalapján (nem kell többé közös fájl szerveret működtetni):



Ezeket a fájlokat azután nem csak desktop módban, de web-en futtatva is ugyanúgy megjeleníti. Hasonlóan járhatunk el a gépek fotójával, az is drag-&-drop technikával vihető be a rendszerbe.



Nagy változás ezen kívül az egyes folyamatok dokumentálása. A korábbi változat Word dokumentumokat használt (.doc), az új azonban HTML5 fájlokat. Hiszen web-en vagyunk, ez az a fájlformátum, ami biztosan megjelenik még egy tableten is. Sőt ezt küldjük tovább azután emailen is, és szemben a korábbi változattal, ami csak plain text, egyszerű szöveg volt, a mostani tartalmazhatja a társaság szimbólumait, web oldalát, címét és mindezt formattáltnak, ahogy a template egy web szerkesztővel, vagy egyszerű szöveg-szerkesztővel előállítottuk. Ez látható bal oldalt.

Végül hatalmas különbség az is, hogy a jelentéseket sem kell többé tárolni, mivel a forrás adatbázisból mindig előállítható – visszamenőleg is. Fájl szinten csatolt dokumentumot tárolunk.

Az új, VISION X10 alapú CMMS minden igényt kielégítő, látványos, korszerű és gyors megoldás a korábbi rendszer kiváltására. Ami ráadásul nem is olyan nagyon bonyolult, léven a program adatbázis szinten a korábbival teljesen kompatibilis.

X10 Verzió 2.21

1. Zip fájl kezelés új lehetőségei

Két új szolgáltatást biztosít a VISION saját Zip menedzsere:

1. Zip-en belüli opcionális könyvtármegadást
2. Tömörítési folyamat (progress) visszacsatolását

Például az alábbi program részlet a DATA.TAB és a DATA.TRN fájlokat csomagolja be a Zip-en belül egy Meter könyvtár alá (korábban a Zip-en belüli könyvtár megadására nem volt lehetőség):

```
zipProgress.Max=2;
fh BackupFile, ZipAdd, "Data\DATA.TRN", "Meter\DATA", zipProgress;
fh BackupFile, ZipAdd, "Data\DATA.TAB", "Meter\DATA", zipProgress;
```

A vastagított rész új:

- a 4-ik paraméterben a belső könyvtár megadása (Meter\),
- valamint az opcionális progress változó (zipProgress).

A fenti példában 2 fájlt csomagolunk be, ezért a progress változó 0-tól 2-ig fog változni. Menet közben az FH parancs visszaadja a vezérlést a kép-programnak, ezért azon egy egyszerű bárgráffal a zipProgress-t megjeleníthetjük. Amennyiben ezt a lebegőpontos változót nem adjuk át az FH parancsnak, ő sem adja vissza a vezérlést, amíg a műveletet be nem fejezi.

Két egyszerű és hasznos módosítás, amivel a programon belül gyakorlatilag bármilyen adatot, fájlokat archiválhatunk. Akár manuálisan, akár programatikusan.

2. Új, hasznos függvények

Számos új függvénnyel bővült a program, amelyek részletes leírását megtalálja az olvasó a VISION Programming Reference Manual nevű interaktív súgójában:

2.1. INI fájl kezelés (pl. VPR.INI paramétereinek kiolvasása, visszaírása):

```
SetVprIni:: procedure (part: shortstring; item: namestring; value: namestring; inifile: optional string)
GetVprIni:: string function (part: shortstring; item: namestring; inifile: optional string)
```

A fenti függvényekkel, nem csak a VPR.INI kezelhető, de az opcionális inifile nevének megadásával bármilyen másik fájl is. Az alábbi példa az SMTP (levelező) paramétereket olvassa ki, teszi módosíthatóvá az alkalmazói programban:

Beolvasás:

```
smtpHost=GetVprIni("SMTP", "Host");
smtpPort=val(GetVprIni("SMTP", "Port"));
smtpSender=GetVprIni("SMTP", "From");
smtpUser=GetVprIni("SMTP", "User");
smtpPass=GetVprIni("SMTP", "Pass");
smtpAuth=GetVprIni("SMTP", "Auth") <> "no";
```

Visszaírás:

```
SetVprIni("SMTP", "User", smtpUser);
stb.
```

2.2. Régi fájlok automatikus törlése

```
MaintainCache:: procedure (filesan: string; timeback: long)
```

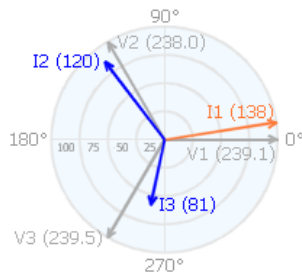
A paranccsal bármilyen könyvtárban tárolt fájljainkat törölhetjük, amennyiben azok régebbiek, mint a *timeback* paraméter által meghatározott idő. Ide pl. 86400-at írva az előző napi adatokat törölhetjük. Hogy pontosan mit, azt mondja meg a *filesan*. Például: [web\charts\files\vt*.html](#)

X10 Verzió 2.20

1. Új chart elemek

Mejelenl egy új chart elem háromfázisú villamos energia mérések megjelenítésére:

Feszültség - áram vektordiagram



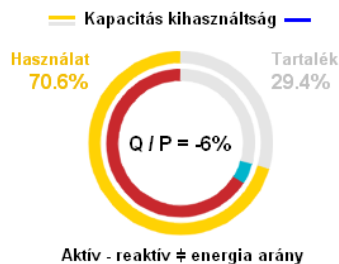
A chart típusa: *crtPhasor*. Ezt kell kiválasztani. Maga a chart 6 vektorral dolgozik, alapdefiníció szerint a 3 fázisú feszültség és a 6 áram, amit egymáshoz képest adott fáziseltolással ábrázol a chart. A szögeket kell megadni a ValueList2-ben.

ValueList: v1,v2,v2, a1,a2,a3

ValueList2: 0, 0, 0, angle1,angle2,angle3

A szöveg fokban kell megadni. A 0 azt jelenti, hogy a szöveget elveti és helyette a 0, 120, 240 – fokkal operál.

A másik érdekes chart a *crtPieart*. Ez valójában két gyűrűdiagram egymásba ágyazva.



Itt is ValueList-től függ, hogy mit ábrázol az első (ValueList) és mit a második (ValueList2) soron. Hogy mennyi elemmel dolgozunk, az a ValueCount-tól függ. A lényeg, hogy ebben az esetben a számok összege lesz a 100%, ehhez képest rajzolja a többi chart elemet.

Végül a chart elemek színe a ColorList-től függ sorrend helyesen.

A példában a teljesítmény felvételt a kapacitás függvényében ábrázoltuk (sárga és szürke), azon belül pedig a második soron a hatásos teljesítmény és a meddő teljesítmény arányát (Q / P).

X10 Verzió 2.18

1. Univerzális adatbázis kezelő update

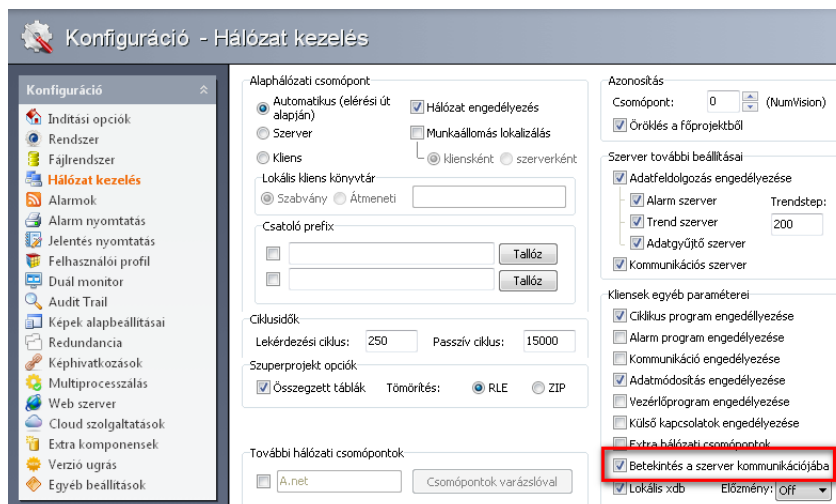
Újabb verzió (8.1.4) került a VISION-be a natív és cloude adatbázis kezelők univerzális rendszeréből. A módosítások a következő adatbázis típusokat érintették (kisebb hibajavítások és performancia növelés történt):

- Az MS-SQL server
- Oracle
- PostgreSQL
- NexusDB (biztonsági protokollal bővült)
- MongoDB (nagy adatmennyiség kezelésével kapcsolatos hibajavítások)
- BigCommerce, Dynamics CRM, FreshBooks, Magento, MailChimp, NetSuite, QuickBooks, Salesforce, SugarCRM, Zoho CRM cloud adatbázisok esetén a kapcsolatfelvétel biztonságosabbá vált

2. Szerver kommunikációs állapot megjelenítése kliensben (2.18)

Ez egy nagyon fontos és hasznos fejlesztés, amivel a kommunikáció állapota megjeleníthetővé vált a kliens programban is. Ez volt eddig az egyik legfontosabb hátrány a VISION szervíz üzemmódban való futtatását tekintve, mivel a szervíz elrejtette előlünk a külső kommunikáció és a hálózati kommunikáció monitorait. Mostantól ez nem hátrány többé: könnyen elérhetjük a kommunikációt a kliens programon keresztül is, ha ugyanazon a server gépen indítjuk el a programot. A kliens osztott memórián keresztül kommunikál a serverrel, ezért szükséges, hogy a "monitorozó" klienst ugyanazon a serveren indítsuk.

A funkció nem automatikus, konfigurálni kell:



Az új opció a kliens egyéb paraméterei között található: **Betekintés a server kommunikációjába**.

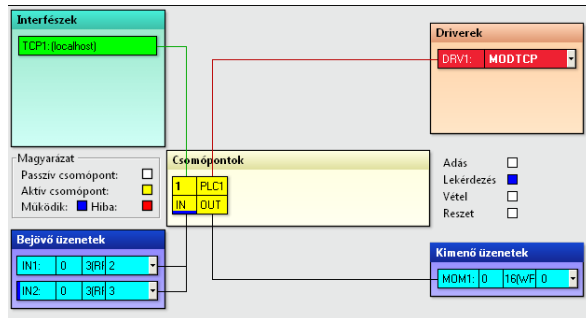
Bár a kliens paraméterei között szerepel, a serveren is be kell kapcsolni, mivel különben a server a kérdéses osztott memóriát nem hozza létre. Ezt az opciót figyelni tehát a server is, közös paraméter.

Fontos, hogy csak abban az esetben használjuk, ha egyébként a VISION kliens önmaga nem kommunikál. Emlékeztetőül, a VISION-ben készíthető olyan osztott kommunikációval bíró rendszer is, ahol a kliens, vagy a kliensek is kommunikálnak hozzá kapcsolódó eszközökkel. Ebben az esetben a kommunikációs képen nem a servert, hanem az aktívan kommunikáló klienst szeretnénk látni, ezért ilyenkor az opció nem használható. Egyébként a *kommunikáció engedélyezése* opciót ki kell kapcsolni. Fontos ezen kívül, hogy a kliens is ugyanazt a kommunikációs taszk listát tartalmazza, mint a server, hisz másképp nem látnánk ugyanazokat a képeket.

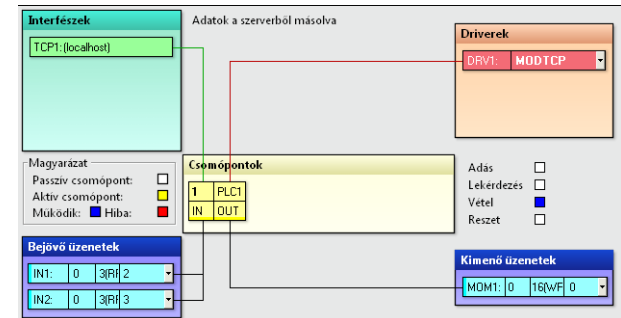
Kikapcsolt *betekintés* mellett szürke lenne az összes driver, bekapcsolt *betekintés* mellett azonban aktívvá válik, plusz megjelenik minden adat, státusz változás, monitor, driver üzenetek stb. – mintha csak a szervert látnánk.

Felvetődik a kérdés, honnan tudjuk, hogy a szervert vagy a klienst látjuk-e, ha egyszer mindkettő ugyanazt jeleníti meg? Az alábbi kép mutatja az árnyalatnyi különbségeket:

Szerver:

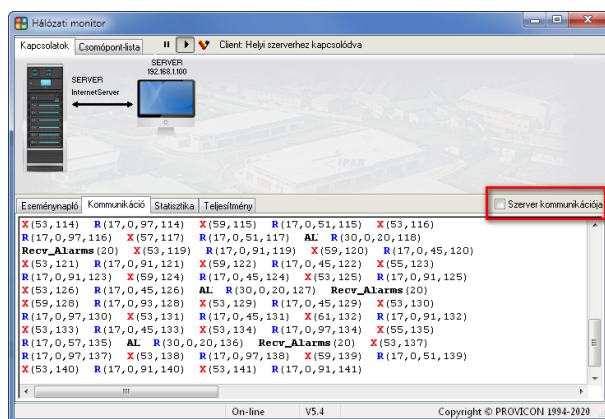


Kliens (betekintő):



Azt lehet látni, hogy a kliens oldalon világosabb árnyalatúak az objektumok, másrészt van egy plusz felirat a kép tetején: *Adatok a szerverből másolva*. Egyébként bárhová kattintunk, ugyanaz történik, azaz megjelenik a szokásos driver monitor vagy a driver üzenetek.

A másik fontos monitor a hálózati:



Van ezen a képen egy új, fontos checkbox ilyenkor: *Szerver kommunikációja*. Ha ezt megnyomjuk, a hálózati monitor nem a kliens hálózati kommunikációját mutatja többé, hanem a szerverét:

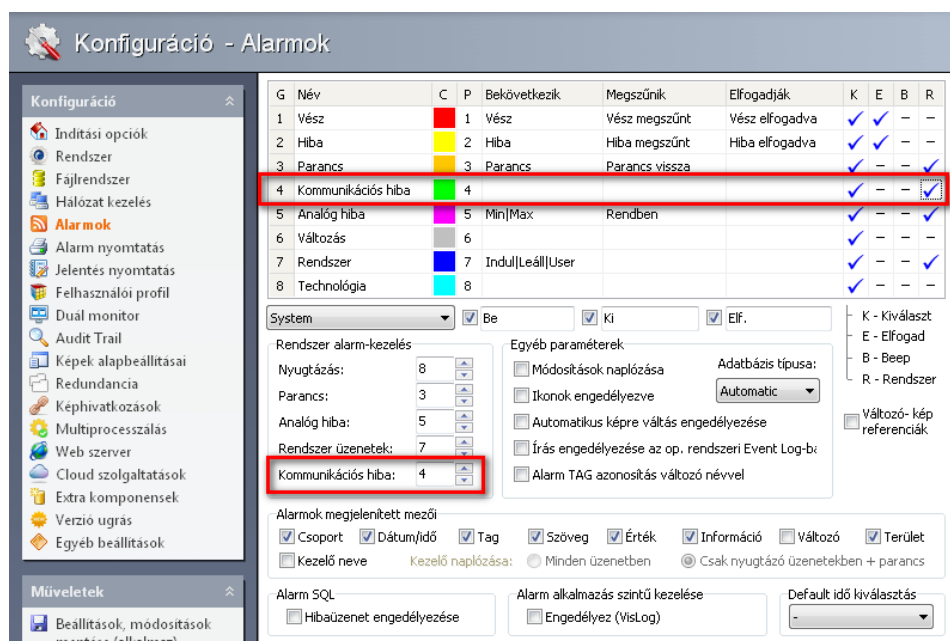


A kép alsó és felső része egyaránt megváltozik: a kommunikáció monitora is és a szerverhez kapcsolódó kliensek képe felül. Mostantól azt látjuk, ahogy a szerver is látja a hozzá kapcsolódó klienseket – beleértve a befigyelőt is. Mi több, az egyes monitorokra kattintva vezérelhetjük is azokat a szokásos módon: leállíthatjuk, felfüggeszthetjük, újraindíthatjuk vagy küldhetünk szöveges üzenetet a felhasználónak.

Végeredményben a teljes szerver menedzsment és monitorozás megoldható a betekintő kliensten keresztül, a VISION szerver szervízként való futtatásának ezzel újabb löketet adva.

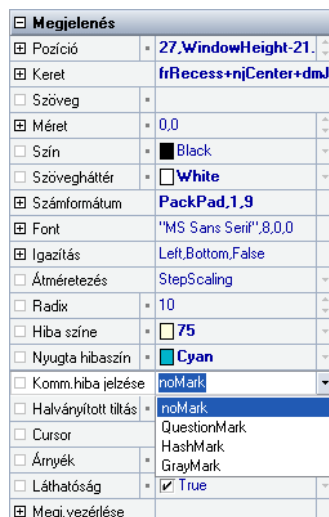
3. Kommunikáció állapotának megjelenítése számkijelzőn (2.18)

Mindenek előtt a rendszer által kezelt események listája bővült egy új lehetőséggel: *kommunikációs hiba*:



A képen látható módon adhatunk az új hibaosztálynak tetszőleges nevet, az **R** oszlopában engedélyezzük azt, majd az adott sorszámot hozzárendeljük a kommunikációs hibához. Ettől kezdve a kommunikációs hiba a változó ALARM mezőjében kiolvasható. Például INTVAR.Alarm.3 formában (bármelyik bit lehet, nem csak a 3-ik). Ha hibás a kapcsolat, ezen a biten 1 van, egyébként 0.

Hogyan használhatjuk ezt az új hibajelzést? A legegyszerűbb módja a számkijelző (NU-objektum):



Itt is van egy új tulajdonság az NU megjelenítést vezérlő sorai között: *Komm. hiba jelzése*. Mint látható, négy lehetőség is van:

- Csak a háttérszín változik
- Kérdőjelek használta
- Hashmark használata
- Szürkítés



Az utolsó háromból bármelyik kombinálható még a háttérszín változtatásával. Nem új, de érdemes itt megismételni, hogy állítható még a nyugtázott hibának a háttérszíne is.

A VISION-ben ennek a kijelzési módnak kisebb a jelentősége, mint más, ma már elavultnak számító SCADA rendszerekben, amelyek az adatok kiolvasását a megjelenített képtől teszik függővé, s ezért először csupa ??? látszik, amikor a képre váltunk, hisz adatok nem lévén, nem is nagyon tehetnek mást. Eléggye szerencsétlen megoldás ez. Kapásból ergonómiai okokból: nagyon csúnya, ha a képen nem látszik semmi, csak hosszú másodpercek múlva. De ennél is fontosabb, kvázi alapelv, hogy a képi megjelenítés nem befolyásolhatja a háttérteszkok működését. S hogy még érthetőbb legyen, miért elavult megoldás ez, képzeljük el a működést Web-en, majd képzeljünk hozzá egy tucat párhuzamos felhasználót. Az adatokat nem a kép miatt kell folyamatosan olvasni, hanem egyébként is (feldolgozás, alarm, trendelés stb.). Amikor pedig képre váltunk, az aktuális adatokat kell mutatnia a programnak, amiket korábban már beolvastunk. A kommunikációs hiba jelzése (???, ###,

szűrkítés) így tényleg csak akkor jelentkezik, ha az adatot nem sikerül a rendszerben frissíteni. Viszont akkor remekül használható.

Még azt tisztáznunk kell, hogy mi számít kommunikációs hibának. A válasz ugyanaz, mint az ún. device változók esetén: a **passzív idő** lejárt. Csomópontonként konfigurálhatjuk, hogy ez az idő mekkora, default értéke 5 másodperc. Amennyiben nem frissül az adat ezen idő alatt, akkor az összes bemenet összes változóját kommunikációs hiba állapotba tesszük (Alarm.3). Egyidőben a device változó deaktiválásával. Ha ráadásul a naplóbejegyzést a device változóhoz rendeltük, a kijelző pont ugyanakkor fog hibára váltani, amikor a naplóban is megjelenik a hiba, s vele egyidejűleg egy tovább lehetséges (új hiba) jelzés a képernyőn, amit megint a device változó vezérel.

4. Trend megszakítása kommunikációs hiba esetén (2.18)

Az előző pontban ismertetett hibaállapot megjelenítésének egy másik lehetséges módja, hogy hiba esetén megszakítjuk a trendgörbét. A trendbe így csak olyan értékeket írunk, ill. jelenítünk meg görbéken, ahol az adatkapcsolat biztosított volt.

Működtetése most is egy konfigurációs opciótól függ, és ugyanúgy szükséges hozzá a kommunikációs hibák rendszer általi kezelése az *Alarmok* konfigurációjánál (ld. előző pontot):

The screenshot shows the 'Konfiguráció - Rendszer' (Configuration - System) window. The left sidebar contains a tree view with 'Konfiguráció' expanded, showing options like 'Indítási opciók', 'Rendszer', 'Fajlrendszer', 'Hálózat kezelés', 'Alarmok', 'Alarm nyomtatás', 'Jelentés nyomtatás', 'Felhasználói profil', 'Dual monitor', 'Audit Trail', 'Képek alapbeállításai', 'Redundancia', 'Képhívatók', 'Multiprocesszálás', 'Web szerver', 'Cloud szolgáltatások', 'Extra komponensek', 'Verzió ugrás', and 'Egyéb beállítások'. The main area displays various system configuration settings. At the bottom of the configuration pane, the option 'Trend mintavételezés megszakítása kommunikációs hiba esetén' (Trend sampling interruption in case of communication error) is highlighted with a red rectangle.

Miután bekapcsoltuk, pont ugyanúgy fogja kihagyni a pontokat a VISION a trendből, mintha leállítottuk, majd újraindítottuk volna a programot.

X10 Verzió 2.17

1. PDF formátumú vektorgrafikus riportok (2.17)

A korábbi változatokhoz képest alaposan megváltozott a riportkészítés. A változtatások összefoglalva a következők:

- Bármilyen méretű riport és ablak kinyomtatható PDF fájlba (nem csak A4-es vagy Letter méretű)
- Az összes szöveg vektoros
- Minden szöveg Unicode, azaz nyelvfüggetlen

A második ponthoz kapcsolódóan: eddig a trendek és a chartok szöveges része bitmapként került a PDF riportba, mostantól ezek is *true type* fontok, azaz vektorosak. Szépséghiba volt ezen kívül a magyar hosszú ő és ű betű megjelenítése, az új verzióban ezek is, mint ahogy minden más nyelv is tökéletesen jelenik meg (orosz, japán, kínai...).

Végül fontos különbség a mérettel kapcsolatos megkötések felszabadítása. Lényegében akármilyen méretű riport vagy ablak kinyomtatható PDF fájlba, azt nyomtatás esetén megfelelően fogja a laphoz igazítani maga a nyomtató program. Ettől függetlenül törekedjünk arra, hogy a lapméret többé-kevésbé továbbra is illeszkedjen a laphoz (A4 fekvő v. állítva), de ez ma már nem olyan fontos, miután nyomtatni se nagyon szoktunk, csak tárolni a PDF fájlt.

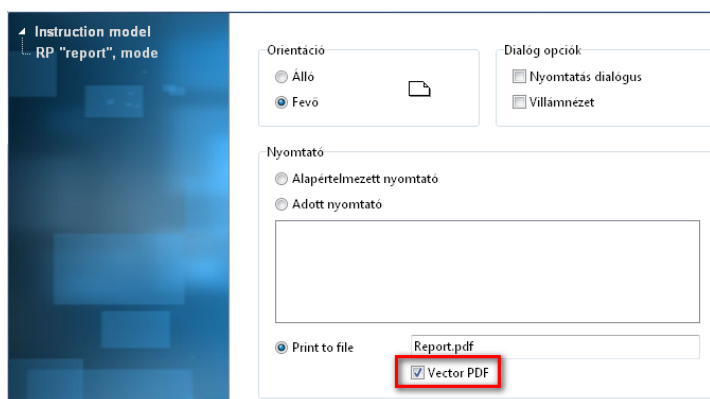
Az új PDF formátum nem automatikus, a *VectorPDF* paraméterhez kötött:

```
rp "TrendReport2", Landscape+VectorPDF, 0, 0, "Report.pdf"
```

Ennek az az oka, hogy a program megőrizze kompatibilitását a korábbi verziókkal.

Természetesen grafikusan, a szervíz varázslóból is kiválasztható:

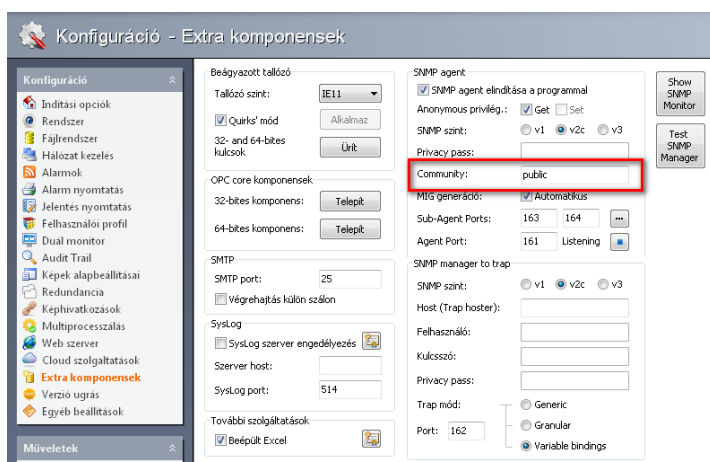
Az itt megmutatott beállítás a fenti parancssort eredményezi – persze begépelve is könnyen előállítható: egyszerűen a *Landscape* vagy *Portrait* kapcsolóhoz hozzá kell adni a *VectorPDF*-et.



2. SNMP community (2.17)

Bővült az SNMP konfigurációs paraméereinek köre a *Community* beállíthatóságával. A paraméter default értéke *"public"*, de ez mostantól változtatható:

A módosításra biztonsági okból volt szükség, hogy nagyobb rendszerekben ne tudjon akárki adatokat lekérdezni a VISION-ből, csak az a komponens, amelyik ismeri ezt a kulcsot.



X10 Verzió 2.16

1. Email küldés új lehetőségei (2.16)

Alapos ráncfelvarráson esett át a VISION Email küldő szolgáltatása. Ezek összefoglalva a következők:

- doMail() függvény bevezetése, ami nem tartalmazza sem a felhasználó nevet, se a jelszavát, se a küldő email címét
- HTML formátumú levél küldése
- Formattált HTML levél készítése
- Többszörös melléklet küldése

Az eddigi *SendMail()* és *MailDialog()* függvények mellé készült a *doMail()*, ami nem tartalmazza tehát a jelszót és a felhasználó nevet. (Eléggé szerencsétlen dolog volt ezeket az információkat korábban a VISION forráskódban tárolni.)

System.doMail < Send email via SMTP server < Vision Programming Reference				
XSDL: doMail: procedure (too: namestring; subject: namestring; attachment: optional namestring; body: optional list of string); Pascal: procedure doMail (too: namestring; subject: namestring; attachment: optional namestring; body: optional list of string); C: void doMail (namestring too, namestring subject, namestring attachment, list of string body);				
Description				
Direct using preconfigured email options				
Parameter	Optional	Var	Type	Description
too	-	-	namestring	Attendee of email
subject	-	-	namestring	Subject of the email
attachment	yes	-	namestring	Attachment
body	yes	-	list of string	Email text in multiple lines

Hol kell azonban megadni a felhasználó nevet és a jelszót, ha nem itt? A VPR.INI fájlban:

```
[SMTP]
Auth=no
Host=mail.t-online.hu
Port=25
From=provicon@provicon.hu
User=provicon@provicon.hu
Pass=Pampam
```

Ez a megoldás azért sokkal szerencsésebb, mert az alkalmazást ezáltal teljesen függetlenítettük a konkrét kiszolgálótól és a forrás szintjén sehol sem tárolunk érzékeny információkat. Emellett rövidebb és áttekinthetőbb is lett a felhasználói program.

Nézzük most meg a HTML formátummal kapcsolatos tudnivalókat

1.1. HTML formátum használata

Ebben az esetben a *Body* nem szövegek sorozata, hanem egy fájlnev "\$" kezdőkérdőjellel kiegészítve:

```
doMail("provicon@provicon.hu", "teszt levél küldése", "", "$Levelszoveg.html");
```

A levél ezután bármilyen összetett HTML lap lehet egyetlen fontos megköötéssel: a HTML kódban található összes bitmapnak (bmp, jpg, gif, png) ugyanabban a könyvtárban kell lennie, mint ahol a HTML fájl van.

Mivel mostantól a szöveg külső fájl, előnyös lehet annak tartalmát konfigurálhatóvá tenni, hisz a VISION VAL nyelvű forrásain keresztül eléggé nehézkes lenne azt manipulálni.

A megoldás a CopyChart eljárás használata:

System.CopyChart < Highcharts < Vision Programming Reference				
XSDL: CopyChart: procedure (chartsour: string; chartdest: string; values: list of namestring); Pascal: procedure CopyChart (chartsour: string; chartdest: string; values: list of namestring); C: void CopyChart (string chartsour, string chartdest, list of namestring values);				
Description				
Copy chart prototype html file with changing list of values inside				
Parameter	Optional	Var	Type	Description
chartsour	-	-	string	Source chart html file
chartdest	-	-	string	Destination chart html file
values	-	-	list of namestring	List of values in form of tex1=value1,tex2=value2

Ennek első és második argumentuma a forrás- és a cél HTML fájl. A forrás van preparálva, a cél a végleges. Mit jelent a preparálás? Egyszerűen @-al kezdődő szövegrészek elhelyezését a szövegben. Például így:

Tisztelt @UserName,

Mellékelten küldjük a @Comment által meghatározott energiahatékonysági riportot.
Adatait itt tekintheti meg: @Dashboard

Tisztelettel,
@Sender

A behelyettesítés a CopyChart() függvény values argumentumában adható meg:

```
CopyChart (MailSource,MailTarget,
           "UserName="+UserName, "Sender="+Organization, "Comment="+repComment,...);
```

Egyszerűen fel kell tehát sorolnunk, hogy milyen nevet, milyen tartalommal kívánunk behelyettesíteni. És ezt felbonthatjuk több argumentumra, mint a fenti példában is látható. Végeredményben akárhány kulcs=érték páros megadható.

1.2. Többszörös melléklet küldése

Ha több mellékletet is csatolni szeretnénk, a fájlok neveit az *attachment* részeként egyszerűen felsorolhatjuk:

```
doMail("provicon@provicon.hu", "teszt levél", "f1.xlsx,f2.pdf", "Hello World!");
```

Ebben a példában 2 fájlt mellékelünk (f1.xlsx és f2.pdf), de lehet nyilván több is. Így is bajunk lehet azonban a felsorolható elemek számával, hisz az adott argumentumot egy sorba kell írni.

Ezen a problémán segít, ha itt is fájlnévet adunk meg a szokásos módon, \$-al bevezetve, mégpedig XDB típusú fájlt. Ez aztán bármely mellékletet tartalmazhat – akár 100-at is.

Magának az XDB fájlnak a fejlécére (mező nevére) nincs megkötés, az lehet bármi - például Files vagy Attachments. A lényeg, hogy alatta x sorban szerepeljen az összes elküldendő melléklet relatív, vagy teljes fájlnévvel.

X10 Verzió 2.15

1. Univerzális adatbázis kezelő update (2.15)

Újabb verzió (8.1.3) került a rendszerbe a natív és cloude adatbázis kezelők univerzális rendszeréből. A legfontosabb változtatások a következők:

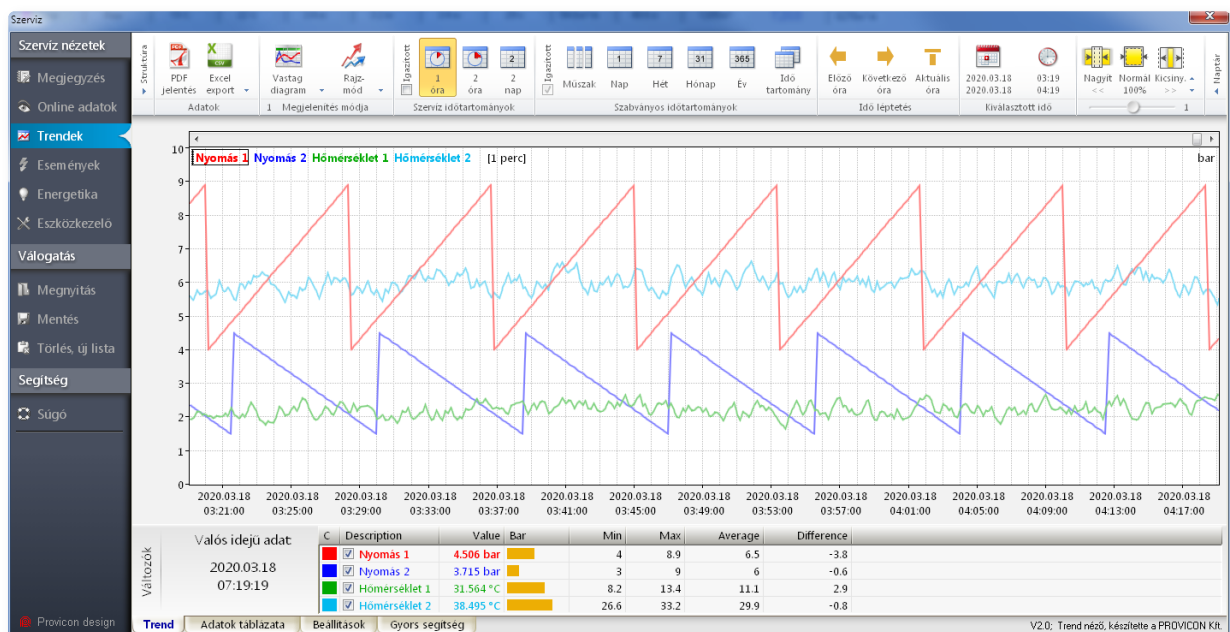
- MS-SQL adatbázisok további performancia javítása
- Ugyanez SAP ASE (Adaptive Serer Enterprise) driver esetén
- Számos apró hiba javítása: MS-SQL TLS 1.2 handshake; ugyanolyan mezőneveket tartalmazó különböző táblákra kiadott SQL query-ik javítása; újrakapcsolódás javítása a szerverhez stb.

A változtatások érintik az MS-SQL, az Oracle, a MySQL, a PostgreSQL, az ASE, az SQLite és a Redshift providereket.

A javítások a klasszikus VISION alkalmazásokat (trend, adatgyűjtés, technológiai adatbázisok) nem befolyásolják. De ettől függetlenül érdemes mindig a legújabb változatot használni.

2. TrendWindow jelstatisztika (2.15)

Egy hasznos módosítás került a TrendWindow plugin-ba: statisztika készítése az épp kirajzolt adatokról:



A statisztika (kép alján) tartalmazza a minimumot, maximumot, átlagot és az utolsó és első minta különbségét, ami számláló típusú integrál adatoknál hasznos, hisz az adott időszak fogyasztását adja vissza. A trend-statisztika működtetéséhez a konfigurációnál 1-oszlopos megjelenítést kell választani:

Változó kiválasztás

Cellat:

Oszlopok: 1 Modulo sorok: 4

Mezők:

☐ Változó név ☐ Custom texts (14)

☒ Szín mező (szín változás)

☒ Értékek a pásztázó kurzornál

☐ Változó név kibővítve a unit rövid nevével

☐ Változó név kibővítve a unit címével

☐ Multiprojekt ☒ Bargráf

X10 Verzió 2.14

1. Kommunikációs driverekkel kapcsolatos változtatások (2.14)

Néhány apró változtatás történt a kommunikációs driverek szintjén. Ez részben a soros vonal kezelését, részben a TCP/IP kommunikációra képes drivereket érintette. A változtatások fő oka a ki-belépési folyamat gyorsítása olyan környezetekben is, ahol az adott soros port vagy IP cím nem elérhető.

1.1. Soros vonalak vizsgálata

A Windows 10 op.rendszer Windows 7-hez képesti jóval lassabb soros vonal kezelése miatt (50ms körüli kapcsolati időt sikerült a Microsoft-nak 3-20 másodpercre növelnie!) ezentúl megvizsgálja a program, hogy az adott COM port létezik-e és hogy kommunikáció céljára egyáltalán felhasználható-e. Nem használható pl. a Bluetooth kapcsolatra szánt soros vonal. Ilyenkor meg sem próbálkozik a program a megnyitással.

Különösen nagyszámú soros vonalat tartalmazó alkalmazásokban jelent ez lényeges változást, amikor a programot saját notebook-on vagy irodai környezetben akarjuk elindítani, ahol az adott kommunikációs környezet nem áll rendelkezésre. Több másodperces felesleges várakozásokat iktathatunk ki ezáltal.

1.2. IP címek vizsgálata

Hasonló okból vezettünk be változtatásokat az S7TCP és a MODTCP driverek esetén. Az adott IP cím elérhetőségét a program ezentúl pingeléssel vizsgálja és csak akkor próbálkozik egyáltalán a kapcsolatfelvétellel, ha az IP cím pingelhető. A MODTCP esetén külön opció ez: PingTest. Az új opció a driver paramétereknél (4-ik paraméter) látható. Azért kellett parametrizálni, mert vannak rendszerek, ahol a ping idő több, mint 200 ms, ami a programnak a ping timeout-ja. Ilyenkor egyszerűbb konnektálni.

Megint ugyanaz a hatás, mint a soros vonalak esetén: amennyiben az IP cím nem elérhető a saját környezetünkben, meg sem próbálkozik a program a konnektálással. Így több másodperces felesleges várakozást iktatunk ki. A probléma itt is a be- kilépési folyamatban, ill. a kommunikációs taszk módosításakor jelentkezett, hiszen amikor a program bezárja a kommunikációt, meg kell várnia, hogy az befejezze az aktuális folyamatokat. Márpedig nem elérhető IP cím esetén az szinte 100%-ban a több másodperces kapcsolatfelvételre vár. A pingeléssel ezt a várakozást iktattuk ki.

Arra az esetre, ha az eszköz mégsem pingelhető, de amúgy képes a kapcsolatfelvételre, a driver előző változata S7TCP_ és MODTCP_ néven rendelkezésre áll.

2. Hálózati kapcsolat biztonságának további növelése (2.14)

A VISION és DFS hálózati kommunikáció, ami a rendkívüli hatékonyság érdekében *delta-frame* tömörítést használ (az előzőleg megküldött adatokhoz képesti különbségeket küldi csak el a program tömörítve), kibővült egy további biztonsági szinttel: amennyiben az egyszerre módosult adat mennyisége meghalad egy bizonyos mértéket (5-15%), akkor a táviratot gyanúsak minősíti és eldobja. A módosítás kompatibilis a rendszer korábbi változataival, vegyes rendszerekben is gond nélkül használható.

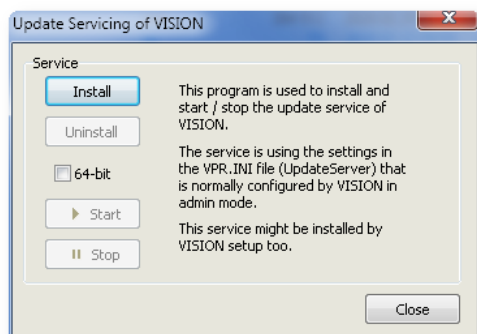
E változtatással 5-re nőtt a VISION és DFS hálózati táviratok biztonságát növelő vizsgálatok száma:

- 32-bites ellenőrző összeg az aktuális táviratról
- 32-bites ellenőrző összes az előzőleg elküldött táviratról (hogy csak akkor XOR-olja az adatokat a program, ha az előző adattábla ellenőrző összege is azonos a távirat-fejben megküldötttel)
- Adás-vételi sorszámok vizsgálata (sorszámozott protokoll)
- Véletlen extra adat validáció a táviraton belül (elhelyezünk a táviratban egy random adatot)
- **Tömeges változtatásokkal szembeni védelem – a mostani módosítás**

X10 Verzió 2.13

1. Update szervíz (2.13)

Az update folyamat további segítése végett bekerült a rendszerbe egy update szervíz szolgáltatás *UpdateService* néven. Ennek installálására vagy egy speciális (a felhasználó által készített) update csomag telepítése során, vagy az *UpdateServicing* nevű segédprogrammal kerül sor:



Az update szervíznek is létezik 32- és 64-bites változata, de maga az installáció ritkán igényli a 64-es változatot, ha csak nem nagyon nagy rendszerek és több tucat projektet tartalmazó szuperprojektek telepítését tervezzük.

Az update szervíznek egyébként az írásvédett környezetek teremtik meg a létjogosultságát. Abban az esetben, ha a normál felhasználói környezetben a fájlok és a telepítési könyvtár is mind írásvédett, de mégis szükség van online update-re, akkor telepíteni kell a szervízt – hisz az emelt szinten hozzáfér az alkalmazás

forrásaihoz, sőt a VISION rendszer elemeihez is.

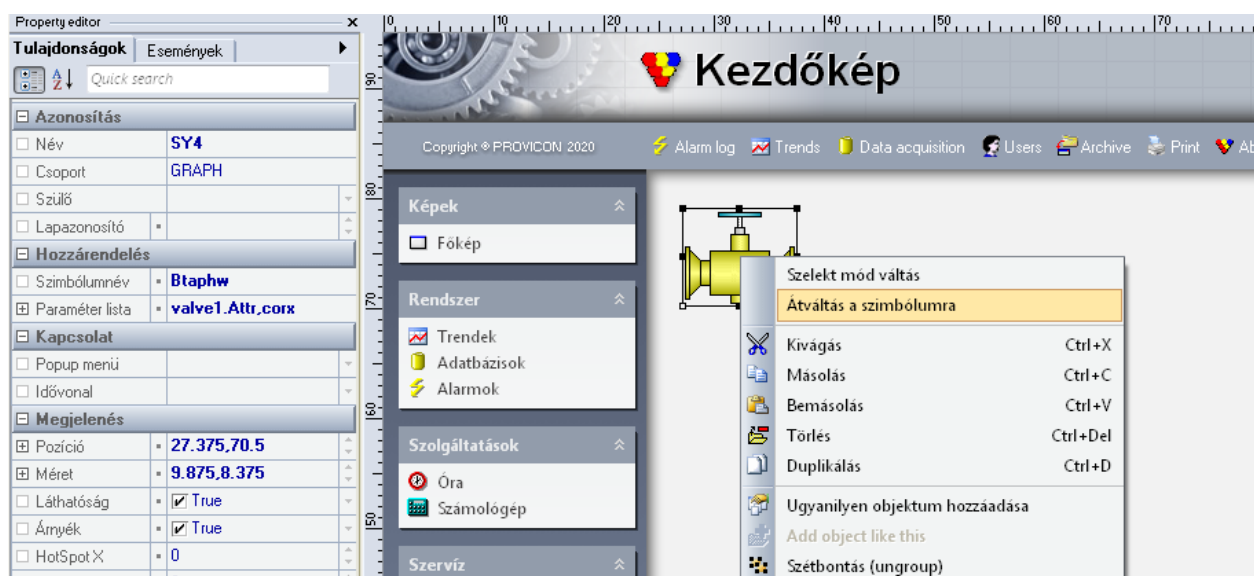
2. Group-ungroup szimbólum management (2.13)

Azok számára lehet kényelmes ez az új szolgáltatás, akik a csoportosítást (*group*) tartják a szimbólum készítés legpraktikusabb módjának. A rendszer számukra is támogatást nyújt, mivel automatikusan elkészíti a szimbólumot. Sőt az *ungroup*, visszabontás is támogatott: ebben az esetben a szimbólum a képen belül kerül kifejtésre és maga a szimbólum hivatkozás törlődik.

Természetesen továbbra is az egyedi szimbólum készítés a javasolt eljárás, hisz ezáltal a szimbólum paraméterezése is megoldott.

2. Átváltás a szimbólumra (2.13)

Ezzel a módosítással könnyen váltogathatunk a kép és a rajta lévő szimbólumok között. Nem kell tehát a nevet megjegyezni és kikeresni a projektmenedzserből, a program ezt automatikusan megoldja:



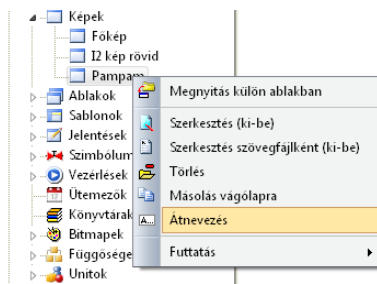
Mint látható, az objektum jobb egérgomb menü bővül az új opcióval.

Egyszerre szerkeszthetjük tehát a képet és rajta a szimbólumokat. A szimbólumon végzett változtatások visszaváltáskor (egér 4-ik gomb, vagy vissza nyíl a toolbox-ban) automatikusan megjelennek a szerkesztett képen.

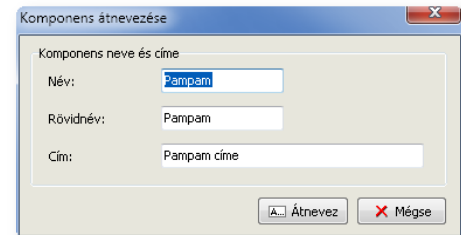
Itt jegyzem meg, hogy a tulajdonságszerkesztőben mostantól különbözőek a default és a módosított sorok: ez utóbbiak vastagítva jelennek meg. Ez is egy apró változtatás.

3. Átnevezés (2.13)

Korábban viszonylag sok lépésben lehetett csak az egyes komponenseket átnevezni, hisz át kellett nevezni magát a fájlt, majd benne átírni a hivatkozást, végül az új nevet módosítani kellett a vpr fájlban is. Mostantól mindez egyetlen művelet:



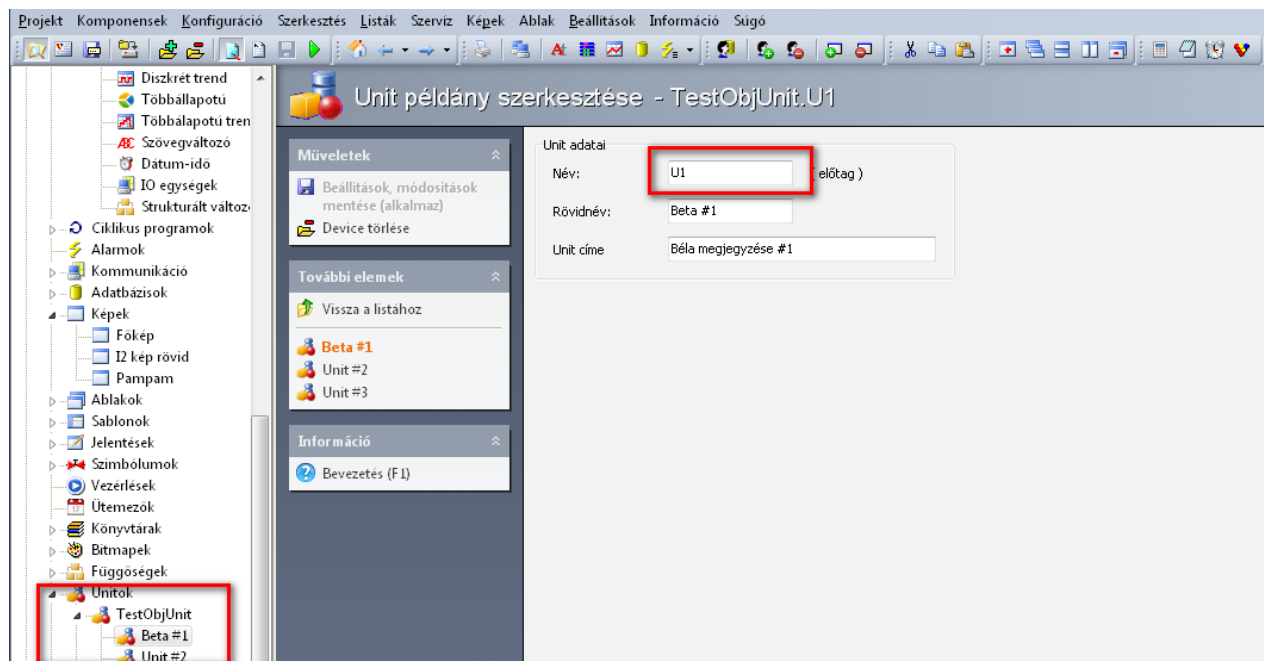
A bal oldalt látható menü (Átnevezés) minden komponensnél megjelenik. Amikor pedig ezt megnyomjuk, a következő képen egyszerűen átírjuk a komponens nevét, rövid hivatkozási nevét és még a címét is (ld. jobb oldalt).



Azonban valami más is történik – a fent leírtakon kívül: **az alkalmazásban található összes hivatkozást is cseréli a program.** Például az exec "képnév" típusú hivatkozásokat.

3.1. Unit átnevezése

Még érdekesebb, hogy akár a unit példányokat is átnevezhetjük – ott is cserélni fogja a hivatkozásokat a program –, és mindezt ráadásul online, anélkül, hogy bármit újra kellene indítani:



X10 Verzió 2.12

1. Backup Management (2.12)

Az 2.11-es Vision X10 és a Vision X9.64.24-es változatok kibővültek egy *backup management* szolgáltatással. Lényege, hogy a forrásfájlokat a program egy alkalmazás könyvtára alatti *_Backup* alkönyvtárba menti automatikusan, amennyiben azok hibátlanok. Vagyis a *_Backup* könyvtár mindig tartalmazza a legutolsó, hibamentes állapotot. A backup management két problémát old meg:

1. **Backup:** ettől kezdve mindig létezik hibamentes mentés az alkalmazás forrásaiból
2. **Rollback:** a hibás programrészeket a program a backup könyvtárból vett hibátlan változattal automatikusan helyettesíti

Ez utóbbi szolgáltatás különösen érdekes. Azt jelenti, hogy még hiba esetén is futtatni fogja a Vision a legutolsó, még hibátlan változatot (képet, szimbólumot, ciklikus programot, alarmot, kommunikációt), azaz egy véletlen hiba miatt sem áll meg sem a kommunikáció, sem az alarm, sem a ciklikus feldolgozás és látható lesz a kép is, annak javított szimbólumaival és sablonaival.

A módosítást a **biztonság további növelése** indokolta, úgy az alkalmazás forrásainak az automatikus mentését, mint a futatott változat hibamentességét illetően. Az alábbi képen pl. az látszik, hogy bár hibás a ciklikus program, az mégsem áll, mivel a módosítást megelőző állapot fut:

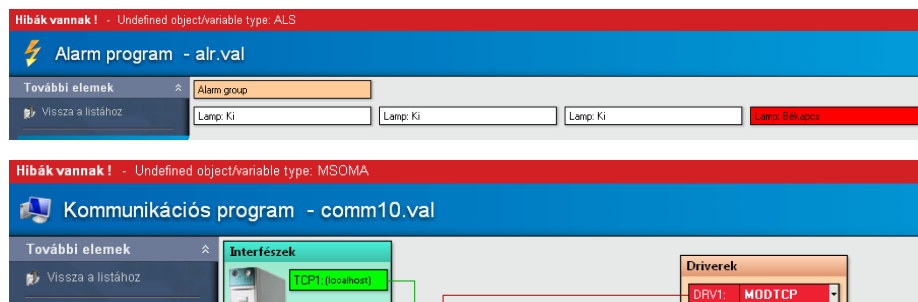
The screenshot shows the Vision X10 software interface. On the left is the main editor window displaying a ladder logic program. A red error message 'Symbol not found: RAT1S' is visible in the editor. On the right is the 'Taszkok real-time monitorozása' (Task real-time monitoring) window. This window contains a table with the following columns: 'Taszk' (Task), 'Modul név' (Module name), 'Fut' (Running), 'Utolsó hiba' (Last error), 'Akt. státusz' (Active status), 'Min', 'Max', 'Processz trend' (Process trend), and 'Processz időszel' (Process time interval). The table lists several tasks, including 'Trend mintavételezés', 'Adatgyűjtés', 'Alarm feldolgozás', 'Ciklikus program', 'Alarm program', 'Kommunikáció', 'Vezérlő program', 'Ütemező', and 'Ciklikus program'. The 'Ciklikus program' row is highlighted in blue, and the 'Utolsó hiba' column for this row shows '<rollback>'. The 'Rollback' button is visible in the bottom right corner of the task monitor window.

A taszk monitor kapcsolódó sora kék és az utolsó hibánál *<rollback>* látszik, jelezvén, hogy a backup verzió fut. Másrészt a program jelzi szerkesztés közben, hogy hol van a hiba a forrásban. Javítás után a kék sor feketére vált, a hibajelzés eltűnik és a backup verzió lecserélődik az új, immár hibátlan változatra.

Azonban nem csak szerkesztés során, de online állapotban is látszik a hiba, mégpedig eléggé hangsúlyozottan (mindig a kép legfelső sorában, piros alapon fehérrel – bármelyik képen vagy taszkon álljunk is):

The screenshot shows the Vision X10 software interface. At the top, there is a red banner with the text 'Hibák vannak! - Symbol not found: RD'. Below the banner is the 'VISION témák' (VISION topics) section, which includes a list of topics and a search bar. The topics listed are: 'Copyright © PROVISION 2019', 'Eseménynapló', 'Trendek', 'Adatbázisok', 'Felhasználók', 'Archiválás', 'Nyomtatás', and 'Névjegyzék'. The search bar is located at the bottom right of the topics section.

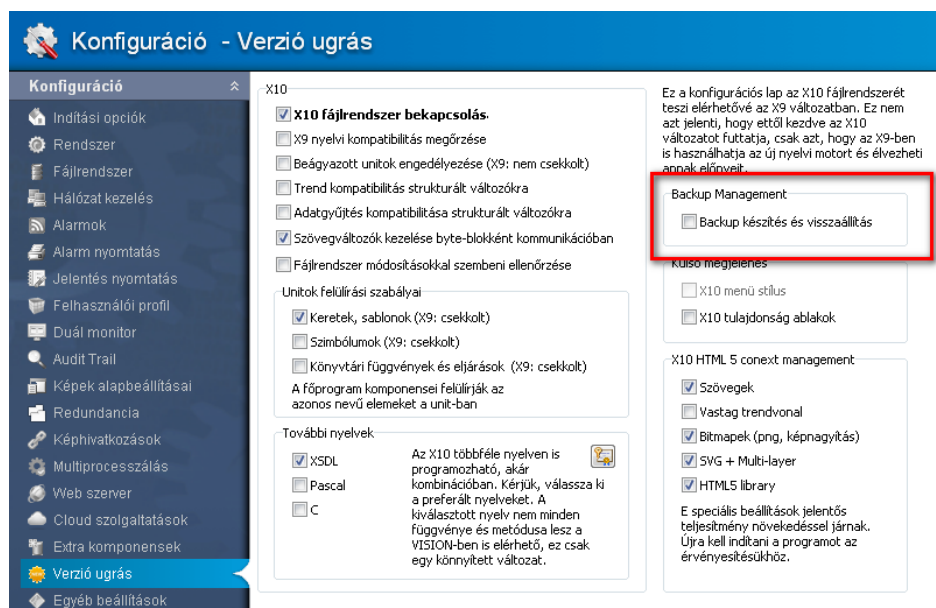
Változott, hangsúlyosabb és egységes lett tehát az online hibajelzés módja és a hibaok megjelenítése:



Bár a *backup management* mindkét Vision változatban működik, Vision X10 esetén indokolt igazán, mivel itt nincsenek objekt fájlok. Vision X9 alatt a program mindaddig a lefordított objekt fájlokat használta, amíg nem tudott a hibátlan forrásból új, jó fordítást készíteni. Ezzel egyfajta rollback (2.pont) szolgáltatást nyújtva. X10 alatt azonban semmit sem látunk egy képből addig, amíg az hibátlan nem lesz. Ezért volt hát szükség a backup management szolgáltatásra. De persze az új megoldás elegánsabb, mint az objekt fájlokra épülő korábbi változat, hiszen forrás szinten biztosítja a visszaállítást.

A lényeg, hogy online módosítások esetén sem áll meg soha egyetlen task sem azért, mert valamit véletlenül elhibáztunk, sőt akkor sem, ha eleve a hibás kóddal indítottuk el a Vision-t. Végeredményben így biztosítva minden korábbinál nagyobb és hatékonyabb hibavédelmet.

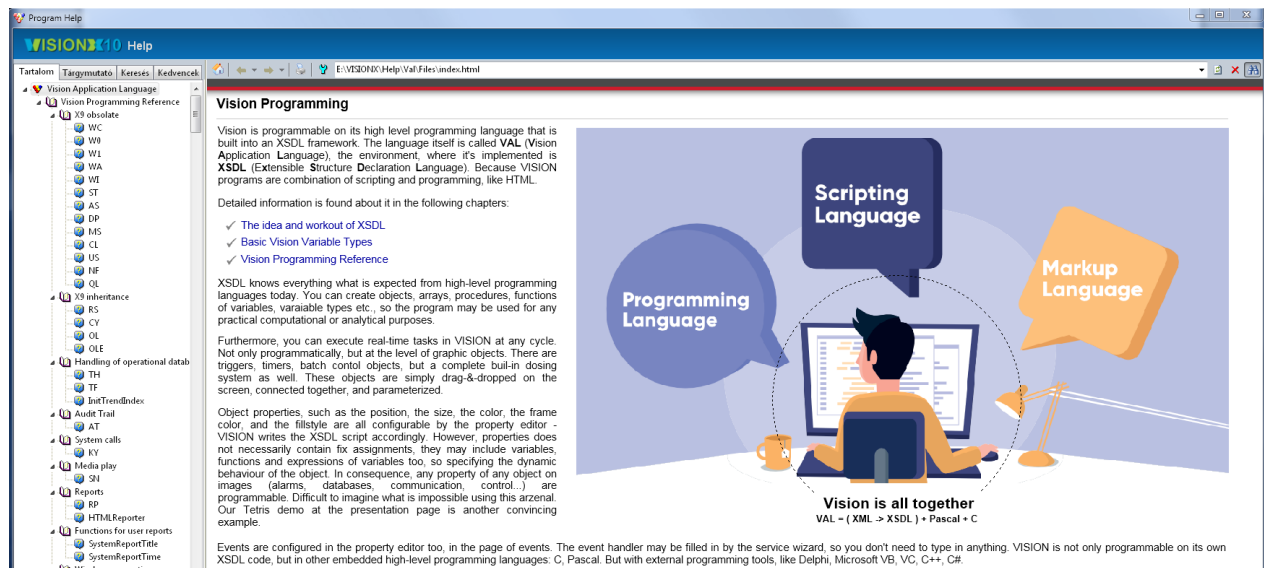
Beállítása a konfigurációban történik:



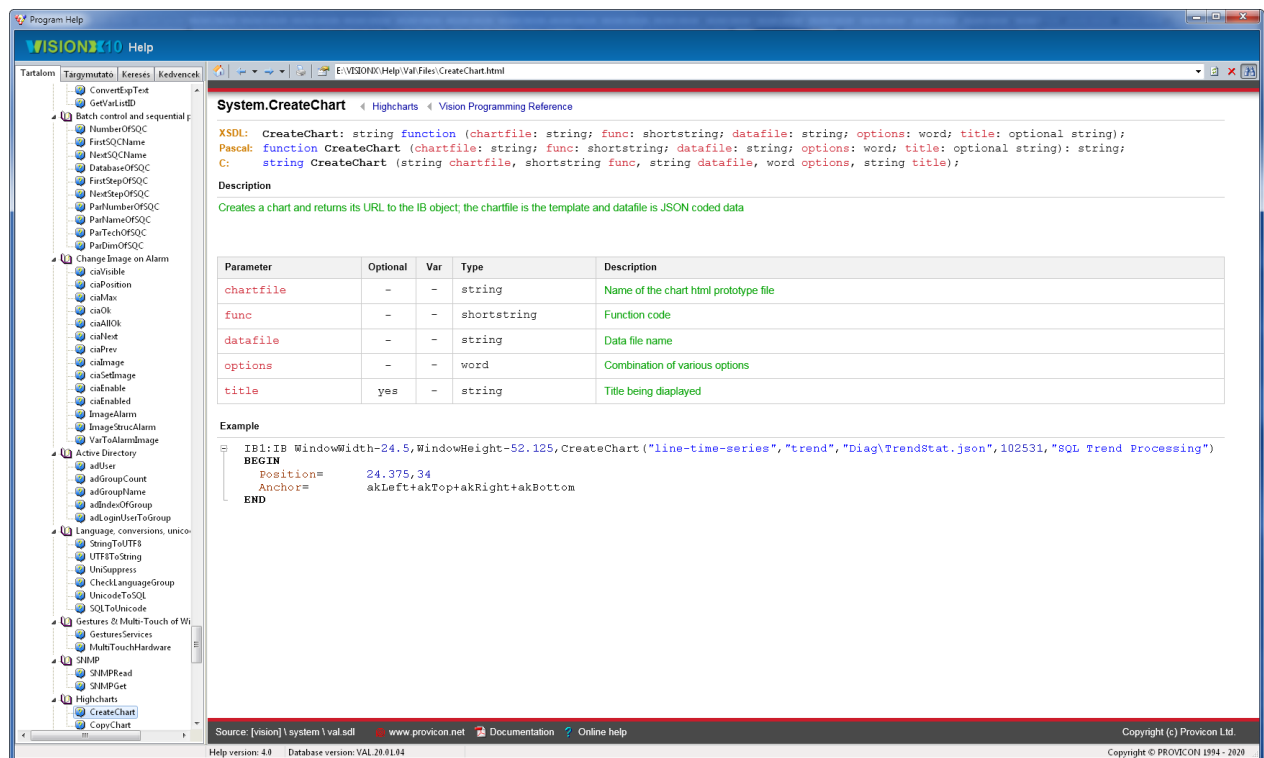
Megjegyzés: a változók nem lehetnek hibásak egy online rendszerben, de minden más igen.

2. Vision Programming Reference Guide (2.12)

Elkészült az utasítások teljes leírása, majd 900 plusz oldal a dokumentációban!

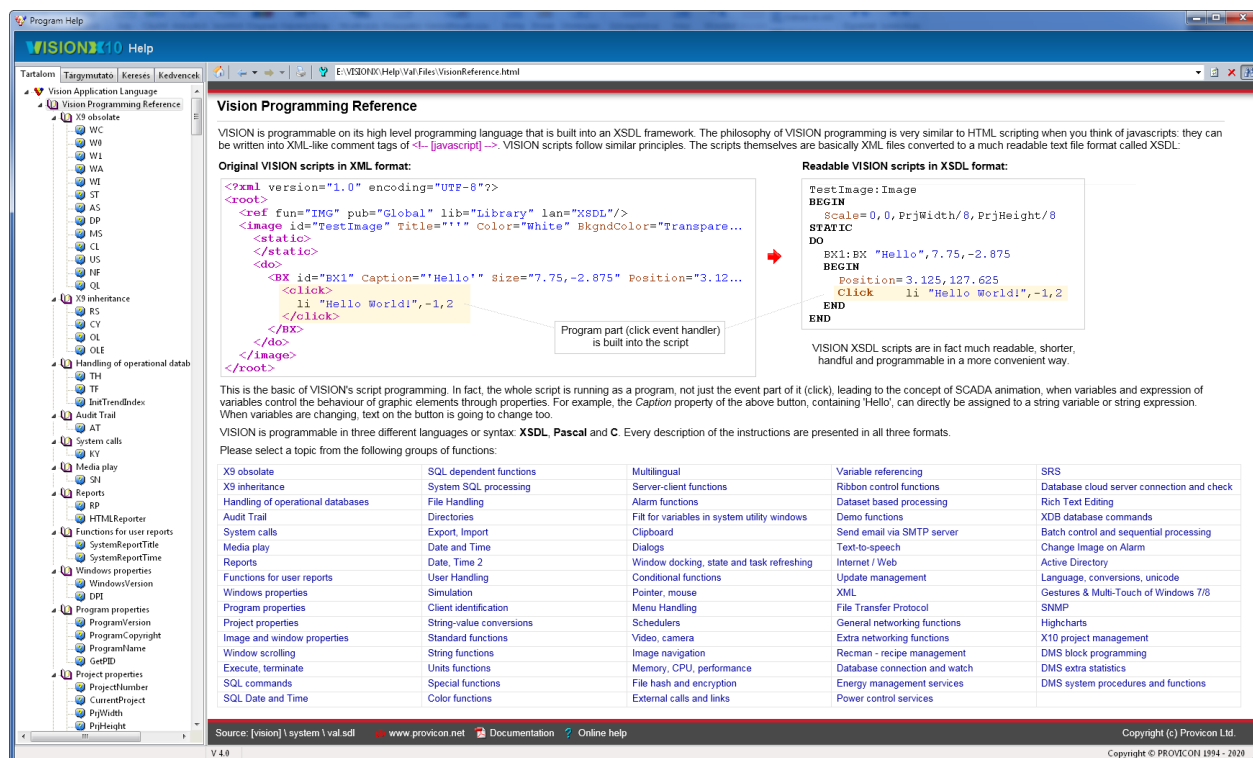


Az egyes utasításokról részletes paraméter listával, leírással, példákkal, s mindezt ráadásul három különböző nyelven, miután a VISION programozható Pascal-ban és C-ben is. Íme:

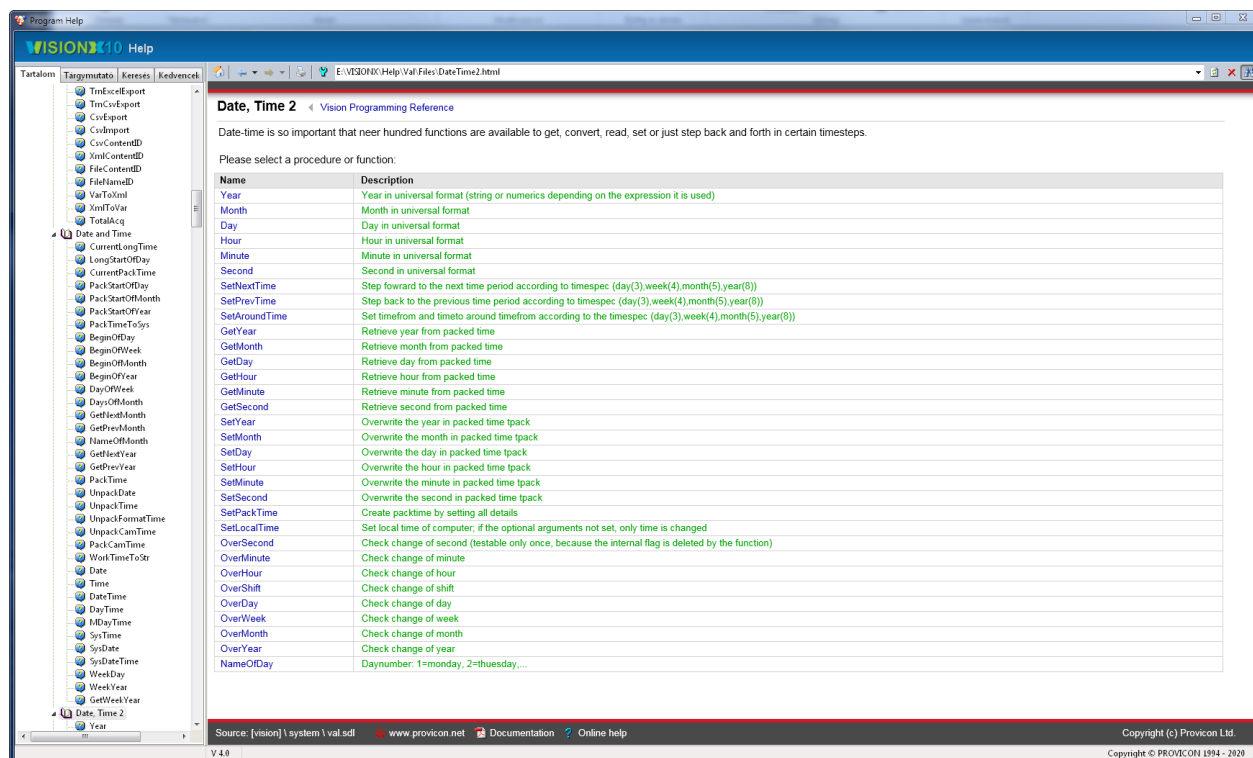


Az utasítások leírása tehát a kibővült súgó részeként érhető el, külön html oldalakon.

Újdonság az is, hogy újracsoportosítottuk az utasításokat a lehető legésszerűbb, a keresést leginkább támogató módon: 80 kategória keletkezett (a kép alján olvasható a lista a Vision Programming Reference oldalon):



Amikor ezek bármelyikét megnyitjuk, a kategóriához tartozó utasítások táblázatát láthatjuk leírással:



Innen lehet továbbmenni a függvény vagy eljárás részletes leírásához, ahogy a legelején mutattam.

Az utasítások sűgön keresztül való elérésének számos előnye van. Például a keresés. A következő képen az látszik, hogy a keresőbe begépeltem az XML szócskát:

VISIONX10 Help

Tartalom Tárgymutató Keresés Kedvencek

Kulcsszó keresése: xml

Topikok: Találat: 11

Cím	Link
Export, Import	ExportImport.html
XmlContentID	XmlContentID.html
FileContentID	FileContentID.html
VarToXml	VarToXml.html
XmlToVar	XmlToVar.html
File hash and encryption	Filehashandencrypti
GetFileCheck	GetFileCheck.html
XML	XML.html
DatabaseToXml	DatabaseToXml.htm
XmlToDatabase	XmlToDatabase.htm
XmlToAcq	XmlToAcq.html

System.XmlContentID

Export, Import Vision Programming Reference

XSDL: XmlContentID: long function (xmlfile: string);
Pascal: function XmlContentID (xmlfile: string): long;
C: long XmlContentID (string xmlfile);

Description

Get unique ID of the XML file according to its content using tag names until the 1st closing tag

Parameter	Optional	Var	Type	Description
xmlfile	-	-	string	XML file name

Alatta láthatóak a találatok, amelyek bármelyikére kattintva megjelenik jobbra a neki megfelelő oldal, benne sárga háttérrel jelölve a keresett szöveget (ez esetben az XML-t).

Tartozik még az utasításokhoz tárgymutató és kedvenc lista is, ahová az általunk leggyakrabban használt utasítások leírásait tehetjük.

A leghasznosabb azonban valószínűleg az interaktív keresés lehetősége. Amikor egy VISION programot nézünk és kíváncsiak vagyunk valamelyik utasításnak a leírására, elég csak megnyomni az F1 billentyűt. Az alábbi példában az *xdbReadInt* függvényre voltam kíváncsi:

Könyvtár - MenuLib.val

További elemek

- Vissza a listához
- MenuLib (MenuLib)**
- Információ
- Bevezetés (F1)

```

76
77 - AfterRightPopup:: procedure (menu: shortstring)
78 begin
79     Globals.lin=         xdbReadInt (1, "Lin", "..\Settings");
80     Globals.col=         xdbReadInt (1, "Col", "..\Settings");
81     Globals.Parts=       xdbReadInt: long function (record: long; fieldname: shortstring; dbname: optional string)
82     Globals.TextColor=   xdbReadInt: long function (record: long; fieldname: shortstring; dbname: optional string)
83     Globals.BarColor=    xdbReadInt (1, "BarColor", "..\Settings");
84     Globals.SmartName=   xdbRead (1, "SmartColor", "..\Settings");
85
86     if menu="ALRACK" then
87         ...

```

read data of given record and fieldname as integer

Amikor az utasítás fölé visszük az egeret, megkapjuk az utasítás prototípusát egy buborékban (ahogy egyébként szerkesztés közben a zárójel után is, mutatván az éppen szerkesztett argumentumot és annak típusát), majd most F1-et gépelek:

System.xdbReadInt

XDB database commands Vision Programming Reference

XSDL: xdbReadInt: long function (record: long; fieldname: shortstring; dbname: optional string);
Pascal: function xdbReadInt (record: long; fieldname: shortstring; dbname: optional string): long;
C: long xdbReadInt (long record, shortstring fieldname, string dbname);

Description

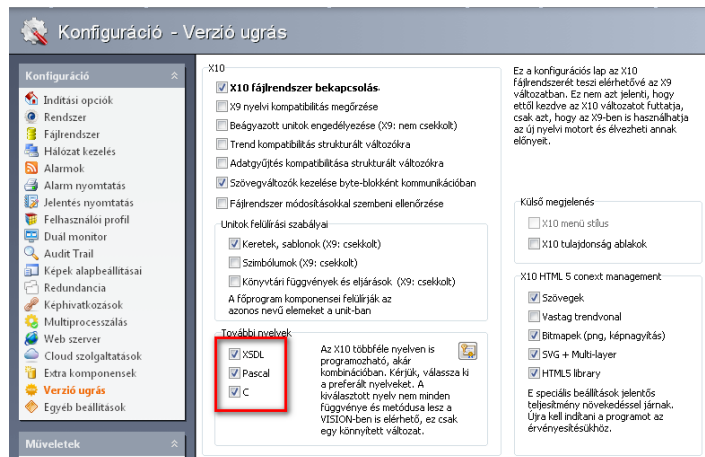
Read data of given record and fieldname as integer

Parameter	Optional	Var	Type	Description
record	-	-	long	Table record number
fieldname	-	-	shortstring	Name of field in database table
dbname	yes	-	string	Database name

X10 Verzió 2.11

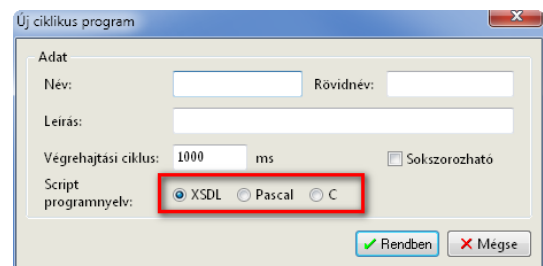
1. Scripting programozás lehetőségei (2.11)

Mint ismeretes, a Vision X10 nem csak XSDL-ben, de Pascal-ban és C-ben is programozható. A használatukhoz mindenképp előtte a nyelvek engedélyezésére van szükség a konfigurációs panelben:



Csak azokat az alternatívákat kínálja fel a program, amikor új képet, szimbólumot, ciklikus programot stb. készítünk, amit itt engedélyezünk. Ezzel kerülhetjük el, hogy ne véletlenül válasszunk C-t mondjuk XSDL helyett és kapjunk hibajelzéseket, miközben azt gondoljuk, hogy XSDL-ben programozunk. Egyszerűen kitilthatjuk, ami nem kell.

Új komponens készítésekor az alábbi dialógus jelenik meg:



Íme, itt választhatjuk ki a scripting programnyelvet, akár képenként, ciklikus programonként mást. De nyilván érdemes törekedni az egységességre. Ha egyszer C-t választottunk, akkor törekedjünk ennek a használatára a projekt későbbi fázisaiban is – mindenhol. A program egyébként megőrzi az itteni választást.

Hangsúlyoznám, hogy a választott nyelv csak a szintaktikát rögzíti, nem a teljes C++ nyelv, vagy Pascal esetén a teljes Delphi lesz ezáltal elérhető! Csak a szokásos nyelvi elemek és a legfontosabb C / Pascal eljárások és függvények. Azokra gondolva, akik jártasak valamelyik nyelvben.

Amennyiben később szeretnénk nyelvet változtatni és még nem tartalmaz a komponens túl sok programot, akkor a forrásfájl legeljén található `#lan` sort írjuk át (default = XSDL):

1	<code>#fun IMG</code>	1	<code>#fun IMG</code>	1	<code>#fun IMG</code>
2	<code>#pub Global</code>	2	<code>#pub Global</code>	2	<code>#pub Global</code>
3	<code>#lib Library</code>	3	<code>#lib Library</code>	3	<code>#lib Library</code>
4	<code>#lan C</code>	4	<code>#lan Pascal</code>	4	<code>#lan XSDL</code>

3.1. Pascal programnyelv

Az XSDL nyelv valójában eddig is nagyon hasonlított a Pascal-ra, ám Pascal-t választva egyezni fog vele. A teljesség igénye nélkül a legfontosabb nyelvi elemek, ill. eltérések az XSDL-től:

- Kulcsszavak használata (**const**, **var**, **type**, **procedure**, **function**)
- Értékadás nem =, hanem := (kettőspont előzi meg az egyenlőség jelet)
- Konstans megadás nem :=, hanem csak = a **const** kulcsó után
- Változó paraméter átadás eljárásnak, függvénynek, képnek, szimbólumnak a szokásos módon a **var** kulcsszó után; skaláris paraméter a szokásos módon
- Az *Xstrings*, *Xdigits* és *Xnumerics* ugyanúgy működik, mint XSDL-ben, mivel ezek valójában pointerok és a Pascal is támogatja ezeket a pointereket
- Szokásos változó típusok (*byte*, *word*, *integer*, *shortint*, *smallint*, *longint*, *int64*, *single*, *double*...)
- CopyStr() helyett copy()
- PosStr() helyett pos()

- Bool helyett boolean
- Típusdeklaráció lehetősége (ez új az XSDL-hez képest)

A Vision X10 Pascal nyelvi implementációja valójában többet enged meg, mint az eredeti nyelv, mert pl. engedi változók deklarációját bárhol a programtörzsön belül is – bár nyilván érdemes törekedni rá, hogy mégis a szokásos helyén, a **var** szekcióban vegyük fel őket. Ugyanez vonatkozik a C nyelv implementációjára is. Elfogadja ezen túlmenően a szokásos = értékadást is a := mellett.

Van azonban pár korlátozás:

- Nem object Pascal és nem Delphi (mivel a Vision nyelve procedurális!)
- Pointer az *Xstrings*, *Xdigits* és *Xnumerics* típusokon kívül nem adható át eljárásnak, csak object (mint a Java-ban)
- Eljáráson belül nem lehet új eljárást deklarálni
- Nincs **With** utasítás
- Nincs **Unit** kulcsszó (se implementation, se interface stb.), mivel a hivatkozások automatikusak
- Csak a // komment használható, a { ... } nem, sem a (*...*)

3.2. C programnyelv

Az implementáció magában foglalja a C nyelv csaknem teljes nyelvi környezetét és a fontosabb C nyelvű függvényeket. A teljesség igénye nélkül a legfontosabb nyelvi elemek:

- Összes C operátor (+, -, /, .%, &, &&, |, ||, ==, !=, >=, <=, <, >, ^ stb.)
- val() helyett atoi() vagy atof()
- str() helyett itoa() vagy ftoa()
- *Compound* hozzárendelés: var+=1; var-=1; var&=1 stb.
- Inkrementálás, dekrementálás var--; var++ (csak postfix!)
- string helyett char[]

Az XSDL-hez képest változott a ^ operátor jelentése, mivel az ott a hatvány (power) számításra szolgált:

- ^ hatvány helyett használandó a C pow() függvénye

Ahogy a Pascal esetében, itt is használhatók a korábbi típusok is (pl. string), a változó deklaráció pedig – ahogy Java-ban – elvégezhető a programtörzsön belül bárhol.

Van azonban pár korlátozás, ill. eltérés:

- Nem C++ (mivel a Vision nyelve procedurális!)
- For cikluson belül az első tagban csak egy értékadás szerepeltethető
- Cím szerint átadott paraméterek esetén (ez felel meg a Pascal var-nak) magára a változóra nem kell *-gal hivatkozni (ez elég buta megoldás a C-ben), csak a nevével, mivel a program tudja, hogy az cím
- Csak a // komment használható, a /* ... */ nem

3.3. Miért van szükség többféle nyelvre, miért kell programozható script?

Azok számára, akik Vision-t használnak, ez eléggé triviális kérdés, de nem minden SCADA, sőt csak elenyésző százalékuk büszkélkedhet olyan megoldásokkal, amely a Vision-ben honos.

Talán érdemes összeszedni a SCADA rendszerekkel szembeni alapvető elvárásokat és igényeket e témával kapcsolatban:

1. A legalapvetőbb igény maga a **script**. Azaz hogy legyen egy olyan **szöveg alapú konfigurációs környezet**, amivel leírhatók a SCADA projektek. Például XML formátumban, mint a legáltalánosabban használt és elterjedt leíró.

Akik olvasták a VISION programnyelvről szóló fejezetet, tudják, hogy a Vision scriptek is XML-ek voltak eredetileg, az olvashatóság kedvéért vannak olyanná alakítva, amit ismerünk. De a konvertálás vissza XML-be könnyen elvégezhető. Hogy mennyire általános tendencia azonban ez a számítástechnika más területein is, érdemes megemlíteni, hogy az Office is XML struktúrákat használ ma már, vagy az Edraw rajzoló program is például, nem is beszélve az SVG szabványról, ami a vektorgrafikus képek XML alapú leírója. A rendszerek fejlesztési tendenciája azonos azzal, amit a Vision-ben már évtizedek óta követünk: egyszerűen ezek a rendszerek életképesek hosszú távon (ld. <http>).

2. A másik alap igény, hogy lehetőleg **minden egységesen leírható legyen script-ekkel**. Vagyis ne legyenek a programban külön bináris állományok, speciális konfigurációs fájlok, se konfigurációs adatbázisok(!) a SCADA alkalmazás meghatározására. És ne legyen kevert a megoldás! A tapasztalat azt mutatja, hogy az ilyen módszert alkalmazó SCADA rendszerek nem életképesek hosszú távon, mivel nem biztosított a kompatibilitás a korábbi változatokkal, ill. csak keszerves konvertálásokkal és célprogramokkal végezhető ez el – már ha a SCADA gyártó foglalkozik e kérdéssel egyáltalán. Gondoljunk csak bele, milyen dinamikusan változik az SQL világ, s hogy vajon kompatibilis-e a mostani rendszerek bármelyike akár egy évtizeddel korábbi elődjével. A teljes rendszerleíró érdekes ezért saját kézben tartani, és mindent ugyanazzal a script-tel leírni.

A Vision az első SCADA rendszer a világon, ami mindent ugyanazon a scripting nyelven ír le már az első pillanattól – változókat, képeket, kommunikációt, adatbázist stb. –, **egységes szintaktikával**.

3. A harmadik alap igény a **programozhatóság**. Ma már nem létezhet SCADA rendszer **konfiguráció szintű programozási képesség** nélkül (hogy a legegyszerűbb esetben változó kifejezését is hozzá lehessen rendelni egy tulajdonsághoz, vagy egy egyszerű feltételes kifejezést, for ciklust és hasonlókat). Ezt többé-kevésbé tudja minden SCADA, de egy részük különálló háttérnyelvet használ (VBA), egy másik részük pedig általános magas szintű program-környezetet (komponens alapú SCADA rendszerek). Ez utóbbi megoldás azonban egyáltalán nem praktikus, mivel ezáltal az egész nyelvi implementációt egy SCADA szállítótól független programgyárhoz rendeljük, akik gyakran változtatják a környezeteiket, és gyakran futnak zsalutkába is. Ez történt a Flash ActionScript-tel, vagy a Microsoft Silverlight-tal. A másik probléma ezekkel, hogy túl bonyolultak; olyan, mintha azt mondanánk: aki SCADA alkalmazást akar, fejlessze ki magának .NET-ben. Ezeket én nem nevezném SCADA-nak, inkább árverésnek. Jobb tehát a programozást függetleníteni a nagyoktól, ahogy tette ezt a Citect a Cicode bevezetésével.

A Vision-ben ez a három alapigény egyszerre kerül kielégítésre. Mégpedig úgy, hogy **az egységes script önmaga programozható**. Ahogy egy html oldalba is beírhatunk JavaScript programot – ugyanúgy a Vision VAL scriptbe is. Ez a legkényelmesebb és legidőállóbb megoldás, mivel a konfiguráció és a programozás egyben van, minden kép és tulajdonság, valamint az animációjukért felelős összes kód ugyanott és ugyanazzal a szintaktikával. A felhasználó pedig rendelkezik minden forrással.

Régi specialitása ez a Vision-nek és megfigyelhető, hogy egyre több hasonló felfogású rendszer születik a világban. Ami azonban új a Vision-ben (X10), hogy a script-be ágyazott programnyelv megválasztható. Így azoknak se kell váltani a Vision program kedvéért, akik Pascal v. C (java, javascript) nyelvű programok írásához szoktak.

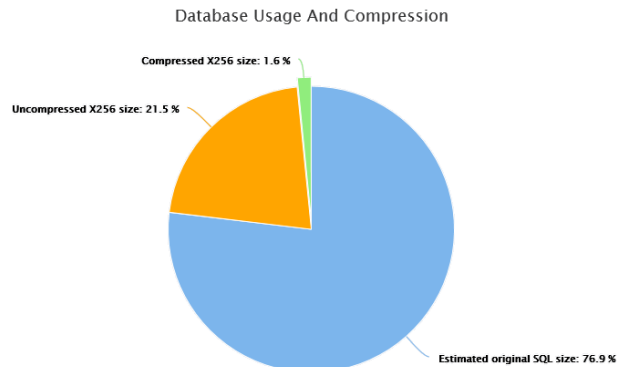
X10 Verzió 2.10

1. X Big Data (2.10)

Mostantól az X Big Data, röviden XBD formátum az alkalmazói programok számára is elérhető (nem csak trendekben, ahogy korábban). Az XBD a VISION saját Big Data formátuma, ahogy a 2.9-es változatban bemutattuk. Ez egy klasszikus Big Data adatbázis, amely azonban kimondottan időbélyeges (*time series*) típusú adatok tárolására lett optimalizálva. Különlegessége, hogy óriási adatbázisok létrehozására és gyors elérésére alkalmas – rendkívüli tömörítettség mellett. Az eddigi legnagyobb projekt 400 milliárd adatot tartalmazott és 6,5 millió mérőváltozót! Érdemes megnézni a róla készült alábbi statisztikát:

Adatbázis statisztika:

Mérőváltozók száma:	6 616 721
Teljes adatmennyiség:	393 737 490 090
Teljes (tömörítetlen) adatbázis méret:	886 371 554 810
Tömörített méret:	52 455 549 306
Tömörítés mértéke:	1,6%
Tárolási hatékonyság:	0,133 byte / adat



Akármilyen hihetetlen, ebben az adatbázisban 1 adat elfér 1 biten és mindössze 1,6% a helyigénye az MS-SQL-el összehasonlítva (tortadiagram).

Az XBD adatbázis a MongoDB-hez hasonlóan no-SQL, vagyis nem relációs. Egy-egy rekord a változók tetszőleges kombinációját tartalmazhatja, vagyis az adatszerkezet a MongoDB-hez hasonlóan időben változhat. Ez néhány alkalmazásban, mint az energiamenedzsment például, kimondottan hasznos. Gondoljunk példaként arra a feladatra, hogy bérlok adatait kell nyilvántartani, de az egyes bérlok időben cserélődnek és a mérők is hozzárendelhetők másként a bérlokhöz (pl. arrébb költözik egy bérlo).

Az XBD-t ugyanúgy kell használni, mint a többi adatbázist: SQL parancsokkal. Az SQL parancsokat a VISION most is önmaga dekódolja és fordítja át az XBD nyelvére – hasonlóan a másik saját SQL kezelőhöz, az XDB-hez (nem tévesztendő össze az XBD-vel, az XDB relációs, az XBD nem).

A deklaráció látható a következő ábrán:

Tulajdonságok

Események

Quick search

DB1

Azonosítás

Név: DB1

Hozzárendelés

Adatbázis típusa: dbBigData

Adatbázis neve: dbNone

Leírás: dbVision

Driver: dbOLE

Kapcsolat: dbBDE

UDL fájl: dbBigData

Autobellépés: dbNone

Lokális Web: dbNone

Path: <ProgDir>\Tmp\XDB\PI00

Szerver: dbNone

Felhasználó név: dbNone

Kulcszó: dbNone

Nyitva: dbNone

HTTP szinkron: dbFalse

Cloud engedélyez: dbTrue

Timeout: db900

Változó UDL: dbNone

DB1: PI00

Leírás: Title of database

Típus: VISION Big Data Engine

Meghajtó: dbNone

DB1: TEST

Változó	Mezőnév	Megjelölt oszlop	Típus	Méret
F10:	P	Power	Float	9.3
F8:	Ptotal	Hatásos telj. totál	Float	9.3
F15:	Q	Reactive power	Float	9.3
F9:	Qtotal	Meddő telj. totál	Float	9.3
F4:	Freq	Frequency	Int	6
F3:	P1	Hatásos telj. 1	Float	9.3
F12:	P2	Hatásos telj. 2	Float	9.3
F17:	P3	Hatásos telj. 3	Float	9.3
F1:	V1	Fesz 1	Int	6
F6:	V2	Fesz 2	Int	6
F7:	V3	Fesz 3	Int	6
F2:	A1	Áram 1	Float	9.3
F11:	A2	Áram 2	Float	9.3
F16:	A3	Áram 3	Float	9.3
F14:	CosPHI1	Cos phi 1	Float	9.3
F5:	CosPHI2	Cos phi 2	Float	9.3
F19:	CosPHI3	Cos phi 3	Float	9.3

A lényeges tulajdonságokat bekereteztem. Meg kell határoznunk az adatbázis típusát (*BigData*), az adatbázis tárolási helyét v. host címét (*Path*) és az időbélyeges tárolás felbon-tását másodpercben (900 = 15 perc).

Ez utóbbi elsőre talán furcsa lehet, hisz az XBD-ben nincs megköve, hogy melyik változót milyen gyakran tároljuk el, az szabadon változhat. Hogy ne legyen azonban túl bonyolult az adatbázis

beállítása és használata, egységes ciklusidejű, ill. felbontású tárolás mellett döntöttünk (később változhat), amit táblánként határozhatunk meg. Az adat hozzáfűzése így egy egyszerű

```
sql "P100.TEST",Append;
```

paranccsal történik, amennyiben a jelen idővel kívánjuk eltárolni az adatokat, vagy az

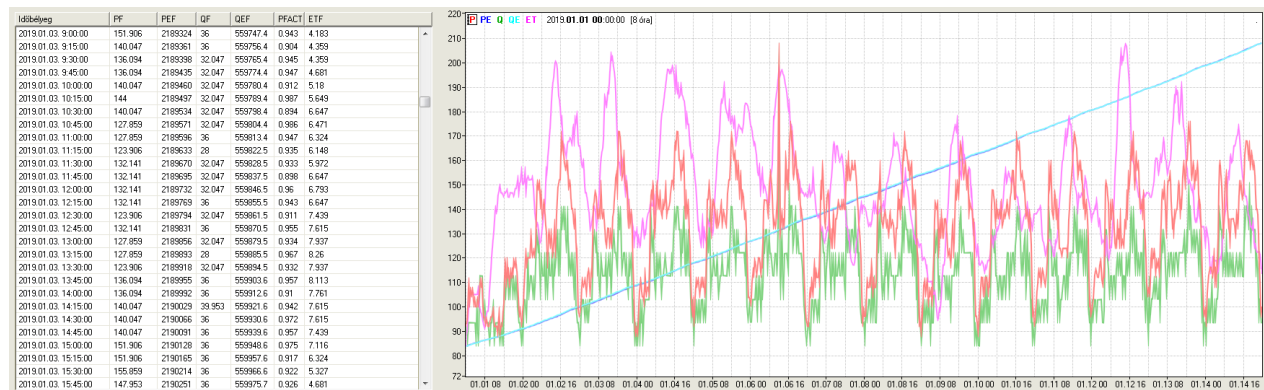
```
sql "P100.TEST",Write,"where TimeStamp=Packtime(...)";
```

paranccsal, amennyiben korábbi időpontban. A tárolás a *TimeStamp* felbontásra kerekített értékével történik (pl. 900 esetén egész 00, 15, 30, 45 perckor). Mint látható, az XBD mindig tartalmaz egy *TimeStamp* virtuális mezőt (akkor is, ha külön nem deklaráljuk), erre hivatkozással tudjuk azután szűrni az adatokat a lekérdezés során. Például így:

```
sql "P100.TEST",Select,"where
TimeStamp>="+SQLDateTime("P100",PackStartOfDay),
"and
TimeStamp<"+SQLDateTime("P100",CurrentPackTime);
```

Az *SQLDateTime()* függvény tehát most is arra használható, hogy a VISION az SQL által megkívánt formába konvertálja számunkra az időfüggvényt. XBD esetén – belül, az SQL utasítás szintjén – ez a *Packtime()*. Tehát nem csak a VISION VAL programban használható a *Packtime* függvény, hanem az XBD adatbázis kezelőben is ezt kell alkalmaznunk, amennyiben nem az *SQLDateTime* parancsot használjuk a célra (erre volt példa a fent bemutatott *Write* parancs). A lényeg, hogy pont ugyanúgy kell kiolvasni az XBD-ből is az adatokat, mint MS-SQL-ből, vagy MySQL-ből – az alkalmazástól az adatbázis kezelést most is függetleníthetjük.

A *select where* feltételrésze tartalmazhat gyakorlatilag mindent, amit más adatbázis kezelőknél megszoktunk (*order by.. desc/asc, min, max, average*, feltételes kifejezések stb.), és hivatkozhatunk az XBD-re közvetlenül trendből (TDX) vagy táblázatból (VX) is:



Ahogy MongoDB esetén, a *create table* itt is hatástalan. A tábla egyszerűen az adatok beszúrával jön létre, és időben akár változhat is a szerkezete. Vision X10-ben működik a [context management](#) is.

Szerver-kliens

Az XBD a fejezeten elején bemutatott formában közvetlenül kezeli az XBD adatcsonkokat. Szerver-kliens rendszerben azonban ez nem célszerű, ilyenkor az XBD-t is úgy használjuk, mint az MS-SQL-t: külön szerver kell hozzá. Ez a szerver program nem más, mint a korábban már ismertetett DBSRV applikáció ill. szervíz. DBSRV használata esetén az adatbázis *Path*-ba nem a fájlcsomók elérési útját kell beírni, hanem a DBSRV host címét (url). Például: <http://192.168.1.100:9090>. A portot a DBSRV oldalon szabadon változtathatjuk és a szerver oldali fájlcsomók, tehát a fizikai adatbázis is ott állítandó be.

Ezután a VISION már klasszikus web socket parancsokkal operál, mint cloud adatbázis kezelés esetén.

X10 Verzió 2.9

1. Natív és cloud adatbázis kezelők (2.9)

A VISION X10-ben hatalmas változás történt, ami az adatbázisok kezelését illeti. Mindenek előtt **bekerült a rendszerbe 14 natív adatbázis kapcsolat**. Ez azt jelenti, hogy a legfontosabb adatbázisokhoz, mint az MS-SQL, a MySQL vagy az Oracle, a jövőben még provider sem kell, mivel a VISION önmaga tartalmazza a providert ezekhez az adatbázis kezelőkhöz. És ez nem minden. A rendszer **kibővült egy no-SQL Big Data adatbázis kezelővel, a MongoDB-vel, valamint egy sor további Cloud adatbázis kezelővel**, amelyek közül az Azure-t és a GCP-t (Google Cloud Platformot) érdemes kiemelni.

A lehetőségeket próbálja összefoglalni a következő ábra.



A VISION X10 6 féle adatbázis alaptípust foglal magában:

- 1. Natív (közvetlen) adatbázis kezelők (MS-SQL, Access, Oracle, MySQL, PostgreSQL, SAP...)**
- 2. OLE DB alapú adatbázis kezelők (MS-SQL, MS-Access, Oracle...)**
- 3. VISION saját adatbázis kezelői (XDB, XBD, DBF, Trend és adatgyűjtés)**
- 4. Cloud adatbázisok és szolgáltatás platformok (BiCommerce, Magento, Azure...)**
- 5. BDE (Borland Database Engine) alapon kapcsolható adatbázisok (DB2, InterBase)**
- 6. Big Data adatbázisok, részben a VISION saját Big Data-ja (XBD), részben a MongoDB**

Az első, a negyedik és a hatodik kategória új, a többi korábban is megvolt. A natív adatbázis kezelők jelentik a legnagyobb előrelépést, hisz függetlenítik a rendszert a Microsoft OLE DB alapú providereitől – amelyekről ráadásul azt mondja a Microsoft, hogy meg fognak szűnni(!) –, másrészt gyorsabbak is azoknál. De például az Access-re nem is volt eddig 64-bites megoldás – ha csak fel nem telepítette valaki az Accdb providert –, mivel e dokumentáció korábbi fejezeteiben leírtak szerint az MDB-hez kapcsolódó Jet 4.0 provider 64-bites változata sajnos hiányzik az op. rendszerekből. Egyszerűen nem készült el. Mostantól viszont megint használhatunk Access-t akár 64-biten is. Sőt egyszerre MDB és Accdb típusú Access adatbázisokhoz is tudunk csatlakozni.

Hasonlóan fontos az Oracle. A Window-ba épített és az ábrán látható Oracle provider elavult, csak a régi szerverekkel tud kommunikálni. A VISION-ba épített változat azonban az összessel. Ráadásul az illesztéshez korábban meg kellett venni az Oracle kliens library-t, kliensenként külön, ami önmagában is horribilis költség és telepítési munka volt. Mostantól ennek is vége.

Fontos még a MySQL-el kapcsolatos problémák áttekintése. Mint ismeretes, a MySQL-t eddig ODBC-n keresztül lehetett csak illeszteni ingyenesen, aminek azonban számos hasfájása volt – kapásból a teljesítménye is elmaradt attól, amit egyébként a MySQL tud. De itt is hasonló volt a helyzet, mint az Oracle-nél: meg kellett venni hozzá és feltelepíteni a MySQL providert, hogy jó minőségű, gyors kapcsolatot létesíthessünk. Mindez mostantól felesleges: a VISION megoldja a problémát, a beépített illesztő gyors, telepíteni semmit nem kell.

S akkor még nem is említettük az SAP illesztésének számos lehetőségét, a PostgreSQL-t, amihez korábban ugyanúgy providert kellett vásárolni és telepíteni, mint a MySQL-hez, az SQLite-ot, amivel saját adatbázis szerverekhez tudunk illeszkedni stb.

A *Cloud* adatbázis kezelők rendszere egészen új dimenziókat nyit meg a VISION felhasználók előtt. Hisz mostantól lényegében a világ összes cloud platformjához tudunk közvetlenül is kapcsolódni. Például az Azure-hoz, vagy a Google Cloud Platformhoz. A legfontosabb azonban talán a MongoDB integrálása, ami a VISION programozók számára további új lehetőségeket kínál, s mindezt ingyenes, gyors, könnyen telepíthető és használható környezetben. A részleteket az oktatásra hagynám.

Összesen majdnem 50-féle adatbázis kezelőhöz kínál az új VISION adatbázis kapcsolatot, számoshoz közvetlenül. **A VISION ezzel a világ összes elterjedt rendszeréhez és szolgáltatásához lett csupán hozzáilleszthető.**

De még ez sem minden! Maga a VISION is tartalmaz egy saját Big Data adatbázis kezelőt, ami gyorsabb még a MongoDB-nél is, amennyiben időbélyeges adatok tárolására használjuk. Továbbá minden itt felsorolt adatbázis kezelő illeszthető VISION Cloud-on keresztül is. Gondoljunk csak bele, mennyire fontos ez! Amennyiben SAP-t, Azure-t vagy akár MongoDB-t használunk távoli adatkapcsolatra – vagyis amikor egy távoli kliens akarja elérni a szerver adatbázisát –, akkor ki vagyunk szolgáltatva a konkrét megoldás adta biztonsági szintnek. Persze a gyártók azt állítják, hogy mindent megtettek a védelem érdekében (néhány hacker most valahol felnevet). Azonban a VISION Cloud adatbázis illesztése továbbra is eléfűzhető az összes natív, OLE és Cloud rendszernek, amivel egyetlen problémára, ugyanannak a Web socketnek a védelmére vezettük vissza a problémát. Ezt úgy védhetjük, autentikálhatjuk, ahogy mi szeretnénk (az egyes adatbázisok saját védelmi rendszerén túl!), csak azokat a portokat nyitjuk ki, amit mi határozzunk meg és csak azt a protokollt engedjük át a routeren, ami ehhez a kapcsolathoz szükséges (HTTP/HTTPS Web Socket). Fontos hangsúlyozni, hogy **pontosan ugyanaz és a konkrét adatbázis típusától teljesen független protokoll működik így a kliens és a szerver között – akármihez is kapcsolódjon az a backend oldalán.**

A VISION historikus adatait is kezelhetjük binárisan, SQL adatbázis szinten OLE DB-n keresztül, és natív drivert használva. Elé pedig ugyanúgy befűzhető a VISION Cloud adatbázis kezelő, mint az összes többi adatbázis esetén. További lehetőség a trendek saját Big Data (XBD) adatbázis kezelőhöz illesztése, ami minden korábbinál nagyobb adatmennyiség kezelésére alkalmas és sokkal gyorsabb is az összes SQL-re, sőt no-SQL-re épülő megoldásnál is.

1.1. Előzetes mérési eredmények

A későbbiekben sokkal részletesebb összehasonlítással fogunk előrukkolni, de azt megállapíthatjuk már az első mérések alapján is, hogy a natív adatbázis kapcsolat gyorsabb. Ez persze várható volt. Azonban MS-SQL esetén nem túl jelentős sajnos a változás, csak kb. 9%. Ez nyilván részben annak köszönhető, hogy már korábban beépült a VISION-be egy komponens, ami az előző (eredeti) ADO technológia kb. [100-szorosát tudta](#), és most ehhez mértük a változást a natív driver alkalmazásával. Valószínűleg itt a vége, ennél gyorsabban MS-SQL-t nem lehet már kezelni.

Az egyes adatbázis kezelőkkel kapcsolatos tudnivalók részletes listája következik:

1.2. VISION adatbázis kezelők

Adatbázis	Leírás	Megjegyzés
XDB-SQL	Az XDB csv jellegű szövegfájlok rendszere, amihez a VISION-t használva szabványos SQL utasításokkal férhetünk hozzá!	legkényelmesebb
XBD (X Big Data)	XDB fájlrendszer alapú no-SQL, Big Data adatbázis elsősorban időbélyeges (hisztorikus), ám időben változó szerkezetű hatalmas adatbázisok kezelésére. Jellemzője, hogy rendkívül gyors és tömör, konkrétan ez a leggyorsabb Big Data adatbázis típus az összes felsoroltból, beleértve a MongoDB-t (ld. összehasonlításunkat)	leggyorsabb!
DBF	Hagyományos dbase 3+ adatbázis kezelő VISION-be ágyazott direkt (natív) és emiatt rendkívül gyors layeren keresztül	elavult
TRN (Trend)	Bináris trend adatbázis hisztorikus adatok rendkívül gyors elérésére. Ez a leggyorsabb megoldás, ám hátránya, hogy több 10 gigabájt adatméret felett nehézkes az adatok archiválása, visszaállítása.	Max néhány Gbyte-ig javasolt a használata
DAT (Data Acquisition)	Bináris adatbázis gyűjtött adatok tárolására. Ugyanazok érvényesek rá, mint a trendekre	

1.3. OLE DB provider alapú adatbázis kezelők

Adatbázis (provider)	Leírás	Megjegyzés
MS-SQL (SQLOLEDB.1)	Microsoft SQL adatbázis kezelő az ún. Express (ingyenes, de korlátozott) változattól az Enterprise, több magot használó nagyobb rendszerekig.	2018-tól nem támogatott! (de lehet, hogy ez változni fog)
MS-Access (Jet.OLEDB.4)	Microsoft Access adatbázis kezelő. A 64-bites változat nem része az op. rendszernek és csak az Accdb-hez van 64-bites támogatás. Ezért 64-bites rendszerben nem ajánljuk.	
Oracle (MSDAORA.1)	Microsoft alapértelmezett Oracle adatbázis provider, mely minden Microsoft op. rendszernek része, de sajnos nem minden Oracle verzióval kompatibilis.	
ODBC (MSDASQL.1)	Provider az ODBC adatforrásoknál beállított kapcsolatokhoz. ODBC szinten érhető el - ezzel a módszerrel - pl. a MySQL is.	
Telepített további providerek (...)	Elvileg minden adatbázis kezelőhöz létesíthető kapcsolat OLE DB-t használva, amihez gyártanak Provider-t. Létezik provider pl. PostgreSQL-hez is.	fizetős!

1.4. BDE (Borland Database Engine)

Adatbázis	Leírás	Megjegyzés
InterBase	Borland adatbázis kezelője. Egyszerűsége miatt még ma is gyakran használják	elavult
DB2	A DB2 az IMB adatbázis kezelő rendszere.	

1.5. Natív VISION X10 adatbázis kapcsolatok

Adatbázis (saját provider)	Leírás	Megjegyzés
MS-SQL (SQL Server)	Microsoft SQL adatbázis kezelő az összes MS-SQL szerverhez közvetlenül, Vision-ből!	gyorsabb, kényelmesebb, jobb, mintha egy további (OLE DB) layeren keresztül kapcsolódnánk
MS-Access (Access)	Microsoft Access, mely támogatja mind az MDB, mind Accdb fájl típusokat és megy (ahogy az összes többi provider is) 32- és 64-bites op. rendszereken is!	
MySQL (MySQL)	Oracle MySQL adatbázis kezelő rendszer. Ez a legelterjedtebb rendszer a világon a Web-es és multiplatform támogatás miatt.	
Oracle	Oracle "nagy" adatbázis kezelőjének az illesztése, beleértve annak összes verzióját: 18c, 12c, 11g, 10g, 9i, 8i, 8.0, Oracle Express Edition 11g and 10g	
PostgreSQL (PostgreSQL)	A PostgreSQL egy nagyteljesítményű, ingyenes relációs adatbázis kezelő rendszer, ami a Berkeley által fejlesztett Ingres utóda. Több, mint 30 év múlttal rendelkezik.	
Advantage (Advantage)	Az Advantage a Clipper, Foxpro, Visual Foxpro adatbázis kezelők összefoglaló neve. Régmúlta visszatekintő, de ma is támogatott megoldások	
SAP - ASE (ASE)	Eredetileg Sybase SQL néven ismert adatbázis kezelő rendszer. Ez az SAP alap adatbázisa, miután a Sybase Corp. az SAP alá betagozódott (ASE = Adaptive Server Enterprise)	
SAP - ADS (ADS)	SAP adatbázis szerver illesztés ADS (Advantage Database Server) alapokon - az Advantage-nél megismert adatbázis típusok SAP rendszeren belüli támogatására	
NexusDB (NexusDB)	A NexusDB Pty Ltd. által - eredetileg TurboPower FlashFile néven - fejlesztett rendszer utóda, előszeretettel használják Embarcadero RAD Studio és .NET környezetekben. Szuper gyors.	
DB2 (DB2)	A DB2 az IBM adatbázis kezelő rendszere. Érdekessége, hogy nem csupán relációs adatbázisokat, de ún. hibrid adatbázisokat is támogat (XML alapú leírókkal).	
Redshift (Redshift)	A Redshift az Amazon adatbázis kezelő rendszere, ami PostgreSQL alapokon (8.0.2) indult, majd továbbfejlesztették internetes felhőszolgáltatásokkal és Big Data analitikával. Az Amazon Redshift jelenleg a legnagyobb felhő alapú áru adatnyilvántartó rendszer a világon.	
SQLite (SQLite)	Az SQL Lite egy kisméretű, nyitott forráskódú, C-ben írt adatbázis kezelő rendszer, ami beépíthető az alkalmazásokba. Ezért nem önálló adatbázis szerverként, hanem valamilyen alkalmazás részeként érhető el és így létesíthető vele Vision-ből kapcsolat.	
InterBase (InterBase)	Borland adatbázis kezelője. Egyszerűsége miatt még ma is gyakran használják, de a jelentősége csökkenő tendenciát mutat.	
Firebird (Firebird)	A Firebird egy nyílt forráskódú adatbázis kezelő rendszer, az Interbase utóda. A szerver egyaránt fut Windows-on, Linux-on és Unix-on is.	

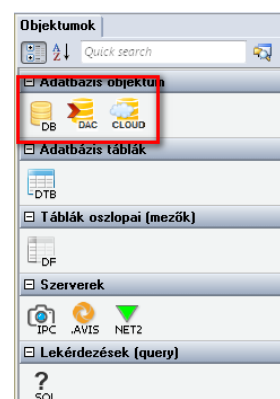
1.6. Cloud adatbázisok és szolgáltatások

Adatbázis (saját provider)	Leírás	Megjegyzés
MongoDB (MongoDB)	A MongoDB a legkedveltebb no-SQL platform és Big Data megoldás. Ennek az a fő oka, hogy nemcsak felhő alapú szolgáltatásként, de lokálisan is telepíthető és használható, és ingyenes	ajánlott platform
Azure (Azure)	Microsoft SQL alapú felhőszolgáltatás. Támogatja a program ezen kívül az Azure MySQL és a PostgreSQL adatbázisokat is. Egyszerre tehát háromféleképpen is hozzáférhetünk az Azurhoz.	A Microsoft skálázható felhőszolgáltatása
BigCommerce (BigCommerce)	A BigCommerce egy amerikai központú, eredetileg ausztrál fejlesztésű és egyre népszerűbb platform Saas célra	Ipari rendszerekben jelenleg nem túl valószínű az igény ezekre az adatbázis kapcsolatokra, de SmartCity alkalmazásokban már elképzelhető
DynamicsCRM (DynamicsCRM)	Microsoft platform elsősorban CRM (Customer Relationship Management) rendszerek támogatására.	
Magento (Magento)	A Magento PHP alapú, nyílt forráskódú platform, mely ráépül a MySQL, ill. MariaDB rendszerekre.	
FreshBooks (Freshbooks)	Elsősorban kis- és közepes vállalkozások részére kínál adatbázis platformot könyvelési és számlázási feladatok ellátására	
MailChimp (Mailchimp)	A Rocket Service Group platformja elsősorban marketing feladatok automatizálására és gyorsítására	
NetSuite (NetSuite)	Kaliforniai központú felhőszolgáltatás platform kis- és közepes vállalkozások számára ERP (vállalatirányítás), CRM (vevői kapcsolattartás) és PSA (szolgáltatás automatizálás) modulokkal.	
QuickBooks (Quickbooks)	Az Intuit Inc. által fejlesztett számlázási rendszer kis- és közepes vállalkozásoknak.	
Salesforce (Salesforce)	Kaliforniai és San Franciscoi központú, rendkívül gyorsan növekvő társaság. Felhőszolgáltatásukat eredetileg a társaság saját CRM és más piaci feladatainak az ellátására tervezték, ma már a publikum számára is hozzáférhető.	
Salesforce MC (SalesforceMC)	A Salesforce Marketing Cloud feladatokat segítő platformja.	
SugarCRM (SugarCRM)	A Sugar webalkalmazás (CRM) köré épülő felhőszolgáltatás.	
ZohoCRM (ZohoCRM)	Elsősorban IoT eszközök számára tervezett platform, de mint a neve is mutatja, itt is vezető szerepet játszik a CRM.	
MariaDB (MairaDB)	A MairaDB az MySQL helyettesítésére szánt relációs adatbázis kezelő rendszer számos többletszolgáltatással. Nyílt forráskódú rendszer	MySQL alternatíva
GCP (GCP)	Google Cloud Platform MySQL és PostgreSQL alapokon. Mobil alkalmazásokkal való együttműködés esetén lehet szerepe.	Már van rá VISION referencia

1.7. Implementáció és használat

Az adatbázis konfigurációs kép két új adatbázis objektummal bővült: DACDB és CloudDB. Az előző a közvetlen elérésű providereket tartalmazza, az utóbbi a Cloud-ot. Egyedül a MongoDB lóg ki a sorból, mivel az mindkét részben szerepelhet (ábra: 6). Ezt végül a Direct Access (DAC) csoportba tettük, hisz valószínűbb, hogy telepített MongoDB installációkkal találkozunk majd. De ez részlet kérdés, hisz DAC-ként is éppúgy beállítható Cloud.

Az új adatbázis objektumokat láthatjuk jobbra. Ezeket pont ugyanúgy kell használni, mint a DB-t és még majdnem pont ugyanúgy is néznek ki. A legfontosabb különbség, hogy az *Adatbázis típusa* itt egy hosszú lista – úgy a közvetlen elérésű, mint a cloud adatbázisok esetén:



Azonosítás

Név: VDB1

Hozzárendelés

Adatbázis típusa: vdbMSSQL

Adatbázis neve: vdbNone

Leírás: vdbAccess

Provider Name: vdbMySQL

Kapcsolat: vdbOracle

UDLX fájl: vdbPostgreSQL

Autobelepés: vdbMongoDB

Lokális Web: vdbAdvantage

Szerver: vdbASE

Felhasználó név: vdbDB2

Kulcsszó: vdbDBF

Nyitva: vdbInterBase

HTTP szinkron: vdbNexusDB

Cloud engedélyez: vdbRedshift

Timeout: 30

Változó UDLX: vdbSQLite

Azonosítás

Név: CDB1

Hozzárendelés

Adatbázis típusa: cdbNone

Adatbázis neve: cdbNone

Leírás: cdbBigCommerce

Provider Name: cdbDynamics

Kapcsolat: cdbFreshBooks

UDLX fájl: cdbMagento

Autobelepés: cdbMailChimp

Lokális Web: cdbNetSuite

Szerver: cdbQuickBooks

Felhasználó név: cdbSalesforce

Kulcsszó: cdbSalesforceMC

Nyitva: cdbSugarCRM

HTTP szinkron: cdbZohoCRM

Cloud engedélyez: ☒ True

Timeout: 30

Változó UDLX: ☐ No

Balra a közvetlen elérésű (DAC) adatbázisok, jobbra a Cloud látszik. Ezek az adatbázisok is *ConnectionString* alapján működnek, tehát pont úgy, ahogy azt az OLE DB esetén megszoktuk. Készült azonban hozzájuk egy új *ConnectionString Builder*. Íme:

Please select the mode you want to connect to databases:

☐ OLE DB based database connections
Microsoft's OLE DB providers to MS-SQL, MS-Access Jet, Oracle, ODBC...

☒ VISION's direct access database components (DAC)
Direct access providers to MS-SQL, Access, MySQL, PostgreSQL, Oracle, SAP, DB2, InterBase...

☐ Cloud databases and services
Cloud database providers to Azure, Google cloud, Dynamics, BigCommerce, Magento, Salesforce...

Click "Forward" or double click on the radio button.

Back Forward

Mint látható, itt három opcióból választhatunk:

Az első továbbra is a már ismert Microsoft adatkapcsolat varázslóra navigál bennünket, a második és a harmadik a közvetlen elérésű

Select DAC Provider: ☒ Apply default port

Provider	Description
SQL Server	Connection to MS-SQL server
Access	Connection to MS-Access, both MDB and Accdb
MySQL	Oracle's MySQL server
Oracle	Oracle SQL: 18c, 12c, 11g, 10g, 9i, 8i, 8.0, Express Edition 10g, 11g
PostgreSQL	PostgreSQL server provider
MongoDB	no-SQL provider to MongoDB, both backend and cloud, Big Data
Advantage	SAP's Advantage Database Server connection, SQL and ISAM
ASE	Connection to SAP's Adaptive Server Enterprise
DB2	Connection to IBM's SQL server
DBF	Standard dBaseIII-dBase10, HiPer-Six, FoxPro, Visual FoxPro
InterBase	Borland's database server, versions up to 2017
NexusDB	SQL server in RAD Studio and .NET environments
Redshift	Connection to Amazon's database server
SQLite	SQL server based on SQLite open source library
ODBC	Open Database Connectivity driver to standard Data Sources

Direct Access

Back Forward Cancel

adatbázisokra, ill. a cloud-ra. Például a közvetlen elérésű adatbázisok listája így néz ki (ld. jobbra). Ebből választjuk ki a megfelelő providert, ahogy a Microsoft OLE DB varázslója esetén, majd megadjuk a szerver IP címét vagy nevét, port címét, a felhasználó nevét, a jelszót és a többi paramétert, amire az adatbázis szervernek szüksége van. Ezek látszanak a következő képeken:

Settings Properties

General

Provider Name: SQL Server
Connection to MS-SQL server

Server:

Port: 1433 ☐ Windows authentication ☒ User authentication

Username:

Password:

Database:

Test Connection ...

Settings Properties

Key	Value
ApplicationIntent	ReadWrite
ApplicationName	
Authentication	auServer
AutoTranslate	True
Encrypt	False
FollowerPartner	False
ForceCreateDatabase	True
IPVersion	IPv4
LockTimeout	2000
MultipleActiveResultSets	False
MultipleConnections	False
NativeClientVersion	ncAuto
PacketSize	4096
Provider	prAuto
TrustServerCertificate	False

A mutatott dialógus önálló program, ami az UDLX kiterjesztésű fájlokhoz asszociált (UDLX.EXE). Ugyanis a kompatibilitás érdekében az UDL alapú adatkapcsolat fölött, azt megtartva, bevezettük az UDLX fájlkiterjesztést – az UDL fájl típus *Data Link Property Editor* asszociációját nem módosítottuk. A VISION-ba épített új UDLX adatkapcsolat varázsló egyben tartalmazza a régi és az új adattípusokat: DAC, Cloud és OLE DB. Ezért volt szükség az első checkboxra, az OLE DB-re, hogy minden egy helyen legyen. Másként fogalmazva, az UDLX fájlba UDL tartalom is beírható (pl. a fájl átnevezésével) és az új varázsló ugyanúgy fogja azt kezelni, módosítani, mint a *Data Link Property Editor*, hisz ilyenkor pont azt hívja meg az UDLX.EXE program is. Viszont az UDLX varázslóban át tudunk váltani a másik két lehetőségre, amikor a Microsoft adatkapcsolat beállítóját bezárjuk.

Az új UDLX egy közönséges szövegfájl azzal a lényeges különbséggel, hogy abban sosem látszik a jelszó (szemben az eléggé bután kivitelezett UDL fájlal). A jelszó kódolása AES 256 titkosítású, azt ember nem fejti vissza, csak a rendszer képes rá, belül. Akkor is, ha a *Connection*-t a VAL nyelvű kódban eltároljuk (ebben is látszott eddig a jelszó):

```

VDB1:DACDB vdbMSSQL, "DAC"
BEGIN
  ProviderName= "SQL Server"
  Connection= "Provider Name=SQL Server;Data Source=localhost;Initial Catalog=Budaors;Port=1438;User ID=sa Password=A8A1892E0358DB4B60597531"
  Server= "localhost"
  Username= "sa"
END

```

A rendszer minden tekintetben kompatibilis elődjével. Például az ALARM.UDL fájl továbbra is működik. Ha azonban az alarmokat egy közvetlen elérésű adatázisra átirányítjuk, akkor a rendszer már egy ALARM.UDLX fájlt fog kimenteni és kezelni, az UDL verziót pedig törli. És ne felejtsük el, az UDLX-ben is elmenthető pontosan ugyanaz az OLE DB connection string, ami korábban az UDL-ben volt, és persze pontosan ugyanúgy is fog működni. Sőt, egy ilyen UDLX kiterjesztésű fájlra kattintva eleve a Microsoft adatkapcsolat beállító panelje jelenik meg, ahogy ezt fentebb már kifejtettem. Azonban ha ilyenkor Cancel-t nyomunk, visszatér a program a varázsló legelső képére, hogy onnan DAC vagy Cloud opciót választva átalakíthassuk a kapcsolatot a gyorsabb és kényelmesebb közvetlen elérésű providerek valamelyikére.

2. Óriás trend fájlok (2.9)

A bináris trend adatbázisok méretére vonatkozóan megszűnt a korábbi 2Gbyte-os korlát! Mostantól több 10- sőt 100 Gbyte is lehet – a 32- és 64-bites rendszerekben egyaránt! Ezzel együtt továbbra sem célszerű túl nagy trend adatbázisokat kreálni a nehézkes másolás és archiválás miatt. A nagyobb rendszereket ezért továbbra is SQL alapokon érdemes kezelni. Sőt akár XBD, Big Data adatbázisokban is tárolhatunk trendeket.

3. SQL context management, context editing (2.9)

Az xdb adatbázisokhoz hasonlóan megvalósult az adatbázisok menüvezérelt, vezetett előállítás is alkalmazói programból. A dolog pont úgy működik, mint az xdb esetén: legépeljük a nevet, utána pont (.) és a program felkínálja az oda beírható lehetőségeket. Ahhoz azonban, hogy ez működjön és kellő szabadságot adjon a programozónak, bevezettünk egy új objektum típust: **TQuery**.

Például az alábbi utasítással létrehozhatunk a programban egy Query-t:

```
Query: TQuery;
```

Ez nem skaláris típus, hanem objektum, nem lehet neki értéket adni egy konstanssal, csak az erre alkalmas *qeury* előállító utasítással:

```
Query = sql ("DATA.Table", Select);
```

Vegyük észre, hogy ez az sql parancs pont ugyanaz az sql parancs, amit eddig eljárásként használtunk: ő is leírható where feltétel résszel, kifejtett sql paranccsal, több sorban – ahogy eddig. Mostantól azonban, mint függvénnyel, értéket is adhatunk vele a TQuery típusú objektumoknak.

Nagyon fontos különbség ez több szempontból is. Az sql mint eljárás, az adatbázisok deklarációjával keletkező két query valamelyikét használta, vagy a select- vagy az exec típusút attól függően, hogy mi volt a konkrét query tartalma (*select * from; update table...*). Valahányszor kiszelektáltunk az adatbázis táblából valamit, az automatikusan öröklődött tovább a **ds** parancsoknak (eredmény halmaz műveletek) és a **VX** táblázat rajzolónak. Függvényként használva azonban nem ez történik. Az sql függvény új query-t hoz létre. Következésképp a benne / általa végrehajtott műveleteknek nem lesz semmi hatása pl. a VX-re sem, ami hatalmas előny a ciklikus taszkban elvégzett sql műveletekre nézve – azokat többé nem kell visszaszelektálni alapállapotba csak azért, hogy a képen ne változzon a táblázat.

A TQuery nem lehet globális, értelemszerűen az mindig lokális változó: csak a taszkban érvényes, ott működik, azon kívül nem.

Nézzük most meg, hogyan hivatkozhatunk a query tartalmára. Íme, pár példa:

```
Query.Datum[12].AsDateTime
Query.ID[12].AsInteger
...
```

A Query második tagja a mezőnév (nem idézőjelek között, hanem így, közvetlenül a programba írva), azután a rekord index szögletes zárójelek között (lehet változó és kifejezés is), vagyis az eredmény halmaz adott sora, és utána az, hogy a mező tartalmát milyen konverziót használva kívánjuk elérni: *AsInteger, AsFloat, AsBool, AsDateTime, AsString*. Ahogy az xdb esetében is láttuk.

Az eredmény (sor) halmaz sorainak a számát a *RecordCount*-ból tudhatjuk meg:

```
Query.RecordCount
```

Innen tudhatjuk, hogy az indexbe írt kifejezés mettől meddig mehet: *0..RecordCount-1*. Ennek alapján már megkonstruálhatjuk a megfelelő lekérdezési ciklust.

A teljesség kedvéért említsem, hogy további query-paramétert használó utasítások is vannak azok számára, akik a hagyományos írásmódot preferálják:

```
sqlDT(Query, 12, "Datum");
sqlInt(Query, 12, "ID");
...
```

A query az első paraméter ezekben az eljárásokban. Itt a teljes lista a VAL nyelv prototípus listájából:

```
sqlInt::      long function (query: pointer; recno: long; field: shortstring) 'read integer cell of dataset; line = recno
sqlFloat::    real function (query: pointer; recno: long; field: shortstring) 'read float cell of the dataset
sqlDT::       long function (query: pointer; recno: long; field: shortstring) 'read datetime cell of the dataset
sqlString::   string function (query: pointer; recno: long; field: shortstring) 'read string cell of the dataset
sqlBool::     bool function (query: pointer; recno: long; field: shortstring) 'read boolean cell of the dataset
sqlMin::      real function (query: pointer; field: shortstring) 'read min of field in the dataset
sqlMax::      real function (query: pointer; field: shortstring) 'read max of field in the dataset
sqlAve::      real function (query: pointer; field: shortstring) 'read average of field in the dataset
sqlSum::      real function (query: pointer; field: shortstring) 'read sum of certain data in the dataset
```

Ezek a **ds** (eredmény halmaz kezelő) utasítások query-alternatívái (*dsInt -> sqlInt, dsFloat -> sqlFloat*), és azért is fontosak, mert X9 alatt csak ők működnek, a kiterjesztett objektum szintaktika a pontokkal csak X10 alatt megy!

X10 Verzió 2.8

1. xdb context management, context editing (2.8)

Az **xdb** adatbázisok kezelése sosem volt olyan könnyű és egyszerű, mint az új VISION X10 rendszerben. Mostantól nem kell bajlódni a nehézkes *xdbRead*, *xdbWrite*, *xdbAppend*, *xdbFF* stb. parancsokkal, helyettük az adott adatbázis mezőelemeire való hivatkozás xdb context hivatkozásait kell írni.

Például az

```
out = xdbRead(1, "Field", "Data");
```

olvasás, ami a *Data* (data.xdb) nevű adatbázis első rekordjából olvassa ki a *Field* oszlop tartalmát, a következő utasítással helyettesíthető:

```
out = xdb.Data.Field[1]
```

Ami túl azon, hogy sokkal áttekinthetőbb és rövidebb is, gépelés közben felkínálja a programozónak a lehetőségeket:

1. Az *xdb*. legépelése után a lehetséges adatbázisokat, benne a *Data*-val
2. Az *xdb.Data*. legépelése után az adatbázis lehetséges mezőit

Még látványosabb a különbség a két módszer között, ha értéket is kívánunk adni az adatbázis elemeinek. Ez az új módszer szerint – például inkrementálva a tartalmat – a következőképpen alakul:

```
xdb.Data.Field[1].AsInteger = xdb.Data.Field[1].AsInteger + 1
```

Egyrészt érdemes észrevenni a *AsInteger* kiegészítést (*AsDate*, *AsTime*, *AsDateTime*, *AsInteger*, *AsBool*, *AsFloat*, *AsString*), ami a mezőt egészzé konvertálja a Java, Delphi, .NET programozói rendszerekhez hasonlóan, másrészt érdemes ezt összevetni az eredeti megoldással. A 2.8 előtti változatban a következőképp kellett volna ezt írunk:

```
xdbWrite(1, xdbReadInt(1, "Field", "Data")+1, "Field", "Data");
```

Mondani sem kell, hogy az előbbi megoldás sokkal áttekinthetőbb ennél, ráadásul minden pillanatban támogatott az *xdb*. context menedzsment által felkínált listákkal. **Tudni előre semmit nem kell, csak mindig kiválasztani a megfelelő elemet a felkínált listából.**

Még ennél is látványosabb eredményt hozott azonban az *xdbFF* (keresés adatbázisban) helyettesítő kódja:

```
xdb.Data.Field.Indexof(123)
```

Ez az utasítás megkeresi a *Data* adatbázis *Field* mezőjének azt az elemét, ahol az érték 123. Az index argumentuma *variant*, írható bele string, bool és változó is, továbbá bármilyen kifejezés.

Azt kell még tudni, hogy az *xdb* után valójában egy relatív path-t, fájllelési utat határozzunk meg. Ha pl. a *Data* adatbázisunk egy *Databases* nevű alkönyvtárban van, akkor a hivatkozás rá:

```
out = xdb.Databases.Data.Field[1]
```

Ha pedig egy szinttel feljebb és az alatt van a *Databases*, akkor

```
out = xdb...Databases.Data.Field[1]
```

Végyük észre, hogy az *xdb* után most három pont (.) van, az első kapcsolja be az *xdb* context menedzsment, a második kettő pedig már az elérési út része: a dupla pont egy könyvtár vissza.

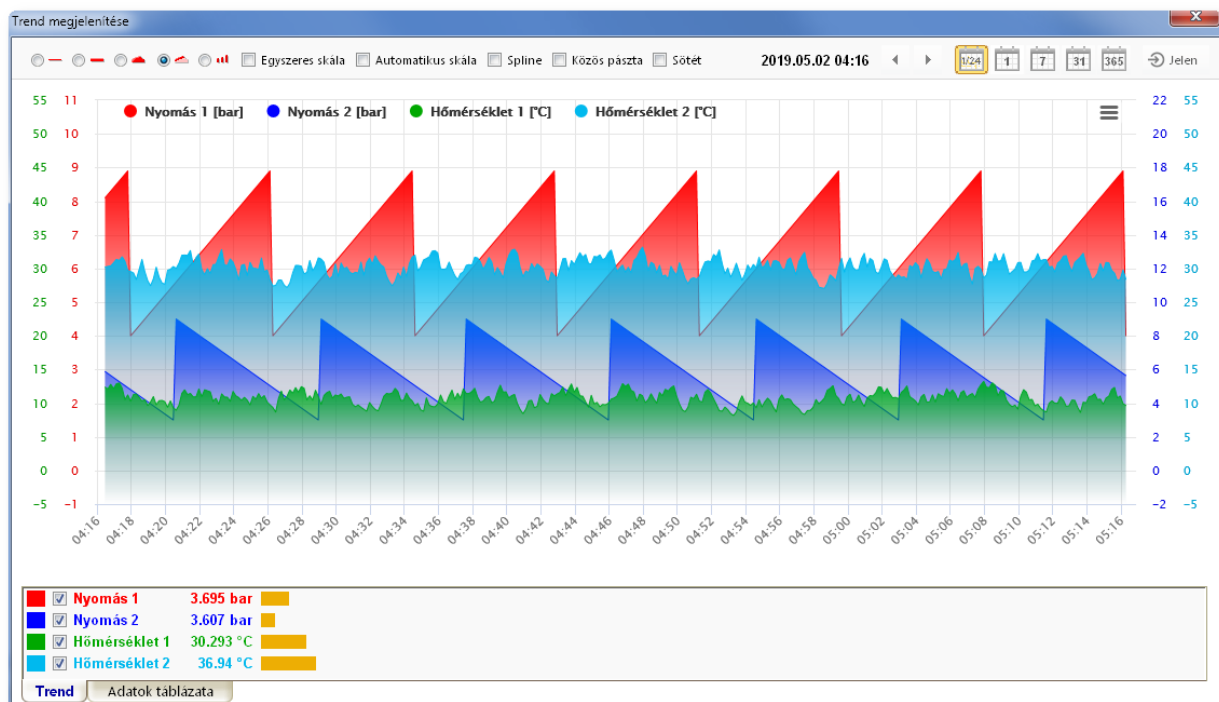
X10 Verzió 2.4

1. Web képes TrendWindow (2.4)

Az új verzió prototípus könyvtárában és az AdvancedCourse alkalmazásban számos változtatás történt a Web képesség érdekében. Változott a *TrendWindow*, *AlarmWindow*, *OnlineWindow* stb. A Web-en is működőképes változatok egységesen w-betűvel vannak kibővítvé (*wTrendWindow*, *wAlarmWindow* stb.). Van ezenkívül egy kimondottan Web-re fejlesztett, speciális TrendWindow is, ami *wTrendWindowLight* nevet kapott, és letezik ennek w nélküli desktop változata is hasonló szolgáltatásokkal. A *light* verzióra a TrendWindow bonyolultsága miatt volt szükség: egyöntetű vélemény, hogy Web-es környezetben az túl összetett. De normál desktop környezetben is felmerült már többször az igény egyszerűsített trend kezelőkre. Az új változatban ezeket a mini plugin-okat fejlesztettük ki, persze továbbra is szigorúan VISION alapokon, VISION alkalmazás fejlesztéssel, hogy bárki igénye szerint módosítani tudja azokat. Nézzük a részleteket.

1.1. Javasolt Web-es TrendWindow

A kép neve *wTrendWindowLight*, utalva ezzel arra, hogy egyszerűsített web-es változatról van szó:



Továbbra is *Varlist* alapon működik a plugin, viszont a trend megjelenítő itt már klasszikus chart, ami a web alatt is pásztázható, nagyítható, szelektálható, sőt kiexportálhatók belőle az adatok XLS vagy CSV formátumban. Ráadásul animálja is önmagát, ha változtatunk az adatokon, a szelekción vagy a skálán. Bizonyos szempontból még többet is tud, mint a desktop megfelelője – végeredményben **full szolgáltatást nyújt a web-en!**

Megjegyzem, mindegyik ablak működik Web-en, az összetett TrendWindow és az összes w nélküli változat is, és megfordítva is igaz, pl. *wTrendWindowLight* is használható desktopon. Vannak azonban olyan megfontolások, amelyek a változtatásokat indokolják, ill. azt, hogy a web-es és desktop változatokat külön kezeljük.

1.2. Különbségek a web-es képeken

Az első fontos különbség, hogy a *Globals* munkaváltozókat a Web-es képeken lokális változókra cseréltük és egységesen egy *TimeStruc* nevű változóstruktúrába szerveztük. Változott tehát a *servicevar.val*. Ez azért volt szükséges, hogy ne legyen áthallás az egyes web session-ök között. Hiszen ha az időtartományt a *Globals.TimeFrom* és *Globals.TimeTo* határozza meg, akkor az egyik session-ben kiválasztott idő látszana a többi Web session-ben is. Ezért kellett őket kikapcsolt Network publicitású változókra cserélni, hisz ismert, hogy ezeknek annyi példánya létezik a szerverben, ahány eltérő session fut rajta. Desktop környezetben azonban továbbra is a *Globals* a javasolt, mivel szuperprojekt környezetben így nem kell külön beállításokkal bajlódni az egyes projektek között!

A másik fontos eltérés a *Varlist* tárolási helyét érinti. A korábbi változatokban egységesen *LocalCommon* könyvtár szerepelt (...User\AppData\Local\Temp...), de web-en ez megint nem jó, hisz az összes session számára ez közös. Ezért ezt a könyvtárnevet mindenhol egységesen *NetSession*-re cseréltük, ami egy VISION függvény, és web-es környezetben a session azonosítónak megfelelő létező alkönyvtár nevét adja vissza. Például: Web\278912\. Amikor a session megszűnik, ez a könyvtár is. Itt tehát eltárolhatók a felhasználó egyedi adatai, így a *Varlist*, vagyis a trendváltozók aktuális listája is.

Azonban ezekhez a módosításokhoz vagy 50 helyen kellett belenyúlni a könyvtári programokba és a képekbe, hisz a *LocalCommon* hivatkozás számos helyen szerepelt. Célszerű azonban ezeket a beállításokat a desktop-ban megőrizni, hogy ne kelljen teljesen eltérő változatokat kezelni. Annak érdekében tehát, hogy továbbra is képesek legyünk desktop és web-es környezetben egyaránt futtatható univerzális alkalmazást fejleszteni, inkább a *NetSession*-t írjuk át a desktop környezetben korábban használt *LocalCommon*-ra, vagy valami másra. Ugyanis ez megtehető a

`NameOfServerSession = "..."`

paranccsal. Még kényelmesebb is, mint a *LocalCommon*, mivel változtatható. Bármit írunk is bele, a Web-es működést az nem befolyásolja, web-en a már említett Web\<session ID>\ stringet adja vissza mindig a *NetSession* függvény. Sőt, ha nem változtatunk rajta egyáltalán, akkor is jó minden környezetben, mivel desktop kliensen a kliens gép nevét tartalmazza és létre is hozza az alkalmazás alatt, szerver környezetben meg az alkalmazás könyvtárára mutat. A szerver és a kliens fájlok továbbra is teljesen eltérő könyvtárakba kerülnek ezáltal desktop környezetben is. Pont ez volt a *NetSession* bevezetésének az oka!

A lényeg, hogy nyugodtan vegyük át az összes módosítást a saját alkalmazásunkba, az jól fog működni a jelentős változtatások ellenére is. De ha teljes kompatibilitásra törekszünk, egyszerűen adjuk ki a

`NameOfServerSession = LocalCommon`

parancsot valahol a ciklikus program elején, a DO rész előtt egyszer, és innen kezdve a *NetSession* azonosná válik a *LocalCommon*-nal desktop környezetben, viszont a Web-en továbbra is úgy működik majd a programunk, ahogy kell.

1.3. Mit kell tenni?

Egyszerűen másoljuk be, vagy szinkronizáljuk az összes plugin-t érintő módosítást az alkalmazásunkba, hiszen az eleve úgy készült, hogy univerzális legyen és jól működjön desktop és web környezetekben is. Ha azonban a saját alkalmazásunkban már szerepeltetünk exec "TrendWindow" típusú hívásokat vagy készítünk LocalCommon+"varlist.xdb" fájlt, ill. hivatkozunk rá, akkor azt cserélni kell NetSession-re:

1. **LocalCommon -> NetSession csere**
2. **"TrendWindow" helyett CTEXT(HTTPClient,"wTrendWindowLight","TrendWindow")**

Ez a legegyszerűbb megoldás. A CTEXT függvény attól függően fog a web-es vagy a desktop alapú TrendWindow-ra hivatkozni, hogy milyen környezetben fut a program.

Hasonló eljárás szükséges az AlarmWindow, OnlinWindow stb. vonatkozásában.

2. Active Directory (2.4)

Az Active Directory (mostantól AD) kezelés nem új funkció a VISION-ben, de számos újdonság van e téren, ezért az egész témakört áttekintem.

Az AD-ből kizárólag a felhasználó kezelést használjuk a VISION-ben. Ez azt jelenti, hogy a felhasználókat nem a VISION-ben tartjuk nyilván, hanem a szervezet saját Active Directory-jában, onnan csak átvesszük azokat. A kérdés ilyenkor az, mi dönti el, hogy kinek, milyen joga van. A válasz: **a jogosultsági osztály, amit az AD csoportjaival azonosítunk**. A programban tehát elég rögzíteni (és egyeztetni a rendszergazdával) az AD csoportokat és azokhoz hozzárendelni a szükséges jogokat:

Felhasználók	Jogosultságok	Szolgáltatások	Opciók	Hatóság	Forrás
	Hozzáférési osztály	Szerkesztés	Konfiguráció	Módosítás	Kulcsátírás
1	Alapjog	✓	✓	✓	✓
2	HelpLibraryUpdaters	–	–	✓	✓
3	Rendszergazdák	✓	✓	✓	✓
4	AD_ADMIN	✓	✓	✓	✓
5	AD_KEZELO	–	–	✓	✓
6	<Új hozzáférési szint>				

Ebben a példában – bár nem feltétlenül szükséges – az összes jogosultsági osztály AD csoportot azonosít (az *Alapjog* kivételével). Ezekhez a csoportokhoz azonban nem szükséges, sőt nem is szabad felhasználót felvenni, hiszen az nem a VISION-ben van nyilvántartva.

A működés ezek után a következő. Amikor a felhasználó belép a szokásos módon, az ő login nevét és jelszavát egy AD hitelesítési eljárásnak vetjük alá, ami ha sikeres, visszaadja a csoportok listáját, amiben az illető user megtalálható. A program pedig összeveti ezt a listát (ami lehet egy nagyon hosszú lista is, hiszen a userek általában nem csak VISION-t kezelnek!) a VISION csoportjaival és ezek közül a legtöbb jogot biztosító csoportot kiválasztja, majd belépteti vele a user-t.

Említettem, hogy felvehetők jogosultsági osztályok az AD csoportokon kívül is, amiket a VISION pontosan ugyanúgy kezel, mint korábban. Egészen pontosan azt csinálja, hogy ha egy user létezik a nyilvántartásban, akkor azzal a csoporttal és jogokkal lépteti be, egyébként az AD-n keresztül. Miért van erre szükség? Egyszerűen azért, hogy az admin jogokat függetleníteni tudjuk a hálózattól, nehogy kizárjuk magunkat a rendszerből.

2.1. AD konfigurációja

Az AD-t a felhasználó konfigurációjánál az opciók alatt találjuk:

The screenshot shows the 'Felhasználók' (Users) configuration window. The 'Active Directory' section is highlighted with a red box. It contains the following options:

- ☒ Active Directory Service
- AD path: WinNT://<domain>/<user>
- ☒ Beléptetés windows felhasználóval (SSO)

Mindenek előtt azt kell tudni, hogy ez egy külön license-t igénylő szolgáltatás, amit rendszer szállítójától vagy a Provicon Kft – től lehet beszerezni. Ezután egyszerűen csak engedélyezni kell. Az esetek 90%-ban ez már elég, mivel az *AD path* alapértelmezett beállítása üres és ezzel a rendszerek többsége működik is. Hogy mikor kell mégis változtatni rajta, a következő pontban ismertetem.

A *Beléptetés Windows felhasználóval* checkbox új. Ennek az a szerepe, hogy a programba a Windows-ba bejelentkezett felhasználót auto-matikusan átvegyük, megúszva ezzel a kétszeres beléptetést. Ez

egy *Signle Sign-on* (SSO) funkció, amit manapság egyre elterjedtebben használnak. Amennyiben tehát a gépbe is átjelentkezik a felhasználó pl. műszakváltás során, akkor a VISION-be már be sem kell lépnie. De nyilván nem használható ez abban az esetben, ha a gépből nem lépnek ki. Ilyenkor egy semleges felhasználót érdemes felvenni (pl. technológia), és utána a VISION-ben bejelentkeztetést kérni. Természetesen nem kell mindenkihez ugyanazt a csoport listát rendelni; pont az a dolog lényege, hogy amennyiben egy másik felhasználóhoz más csoportok tartoznak, akkor azok listáját vetjük össze és adott esetben magasabb v. alacsonyabb szinten léptetjük be. Például az adminisztrátori joggal felruházott személy is bejelentkezhet a diszpécseri VISION-be, amennyiben a rendszergazda rendelt hozzá VISION által is értelmezett adminisztrátori csoportot. Nagyon kevés esetben van ez alól kivétel, de ehhez már az *AD path* fogalmát is meg kell ismerni.

2.2. AD path

Mint említettem, legtöbbször ezzel nem kell törődni. Előfordulhat azonban, hogy a csoportlista mindig a Windows-ba beléptett felhasználó csoportjait adja vissza, hiába jelentkezőnk át. Nem tudunk tehát magasabb v. alacsonyabb szintet kérni (bár az legtöbbször nem is kell, mivel az admin jogra mindenképp érdemes egy saját osztályt fenntartani). Ezen a problémán a

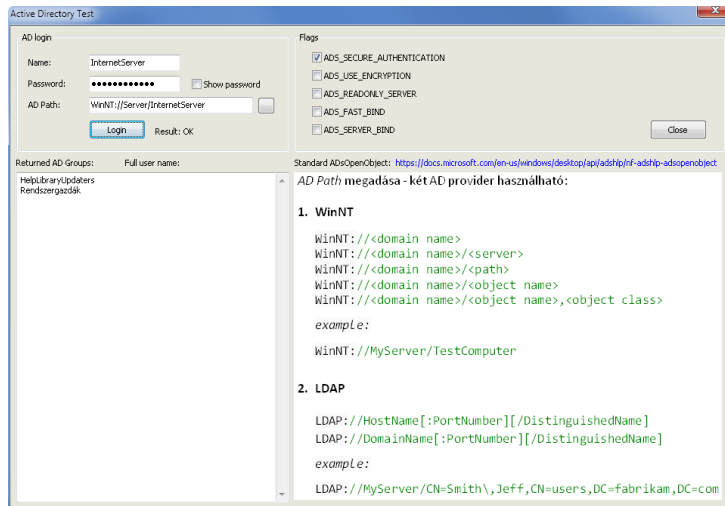
`WinNT://<domain>/<user>`

beállítás segít. Ezzel azt érjük el, hogy a rendszer az AD path-ban a felhasználó nevét majd cserélgeti, mivel a *<user>* az éppen belépő felhasználó login nevére utal. Más esetekben a rendszergazdától megtudhatjuk annak az AD objektumnak a nevét, ami ide be kell írni (nem csak user név adható itt meg, de szerver, objektum név és objektum név, osztály kombináció is).

Előfordulhat nagy ritkán, hogy az AD domain nevét nem tudja megállapítani a program. Azonban mi tudjuk a pontos nevet vagy IP címet. Akkor azt írjuk be a *<domain>* helyére:

`WinNT://babolna.domain.hu/<user>`

A leggyakoribb beállításokat a program mind felkínálja, ha az *AD path* melletti gombot nyomkodjuk. Sajnos az esetek egy részében ki kell próbálni több lehetőséget is, mire a legcélravezetőbbet megtaláljuk. (Zárójelben jegyzem meg, hogy erről elsősorban a WinNT provider tehet, mivel a leírás szerint az képtelen belelátni a többszintű AD objektum struktúrákba; ugyanezen leírások ilyenkor az LDAP provider használatát javasolják). Na ennek a kísérletezésnek a megkönnyítésére készült egy külön kis teszt program *ADtest.exe* néven (Bin-en megtalálható):



Ezzel ki tudjuk próbálni a bejelentkezést és az egyes felhasználókhöz tartozó csoportlistát is megtekinthetjük. Gondoljunk bele, milyen fontos ez abban az esetben, ha valamelyik user nem tud bejelentkezni és ránk mutogatnak, hogy állítólag a csoportot felvették. Na akkor itt megnézhetjük, hogy igazuk van-e.

A program ad némi segítséget a kitöltéshez a jobb alsó sarokban és egy linket a Microsoft megfelelő dokumentációjára. A tipikus LDAP beállításokra is itt találunk példákat.

2.3. AD csoportok kezelése programból

Sajnos van még egy apró probléma: mi van akkor, ha a felhasználót több csoportba is felvették és ezek mind szerepelnek a VISION-ben is. Vagyis nem egy-egy értelmű a felhasználó-csoport összerendelés. Erre is van megoldás: két új függvénnyel lekérdezhetjük az AD csoportokat:

```
adGroupCount::      integer function
adGroupName::      string  function (index: integer)
```

sőt van még egy kényelmi funkció:

```
adIndexOfGroup::    integer function (groupname: string)
```

Például abban az esetben, ha a felhasználó 2 csoportban van, akkor az első függvény 2-t ad vissza (0-át, ha egy csoportban sincs), a második pedig index=1 és index=2 értékekre visszaadja a megfelelő csoportok neveit. A harmadik függvénnyel azt tudjuk lekérdezni, hogy ismert csoportnak mi az indexe (index=0, ha nincs olyan beléptetett csoport).

Ezután a programban felhasználhatjuk ezeket az információkat bizonyos menüpontok engedélyezésére.

Végül lehetőség van arra is, hogy az egyes csoportok között átjelentkeztessük a felhasználót:

```
adLoginUserToGroup:: bool function (index: integer)
```

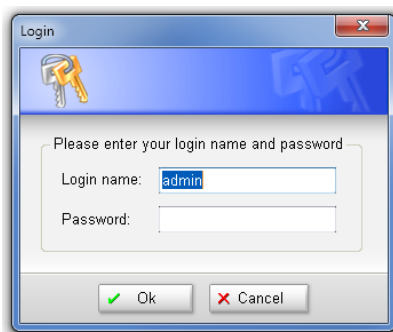
Felhívjuk a figyelmet arra, hogy az Active Directory beállítások csak SQL-be vagy XDB-be átirányított felhasználói adatbázis esetén működnek, mivel a fenti új mezőket a bináris USER.DAT leíróba nem lehetett betenni.

Egyébként, normál beléptetés és több csoport esetén a bejelentkezésre az a szabály, hogy a több jogot biztosító csoportot választjuk ki.

2.4. AD a Web-en

Fontos kiegészítés, hogy az egész AD kezelés Web alatt is tökéletesen működik. Csak azt kell tudni, hogy a tallózó oldali bejelentkezés valójában a Web kiszolgálót tartalmazó szerveren történik meg és ott is tartja nyilván a VISION a felhasználók adatait és teljes csoportlistáját, akár 500 különbözöt, és nem a tallózóban! S persze az előző pontban ismertetett függvények is felhasználótól függő eredményt adnak a lokális kliens példányokban.

Itt kell azonban figyelmeztetni az *AD path* kapcsán leírtakra. A Web kiszolgáló ui. már végképp nem függhet a szerverre bejelentkezett felhasználótól – különösen annak fényében, hogy azon valószínűleg senki sem jelentkezett be, lévén a program szervíz módban fut. Tehát a */<user>* típusú WinNT path mindenképp javasolt, nagyobb organizáció és strukturált AD esetén pedig az LDAP szerver használata.



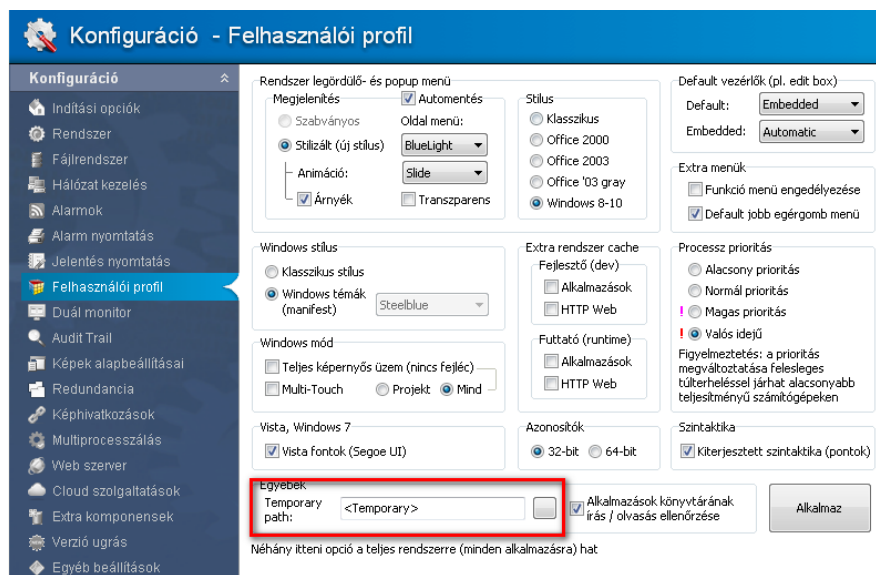
A Web oldali automatikus bejelentkezésre az AD-ba (SSO) jelenleg nincs megoldás, de dolgozunk rajta. Addig is a felhasználót be kell léptetni például ezzel a csinos többnyelvű LoginWindow ablakkal, ami az AdvanceCoursed alkalmazásban található:

X10 Verzió 2.3

1. Csak írható telepítés és alkalmazás (2.3)

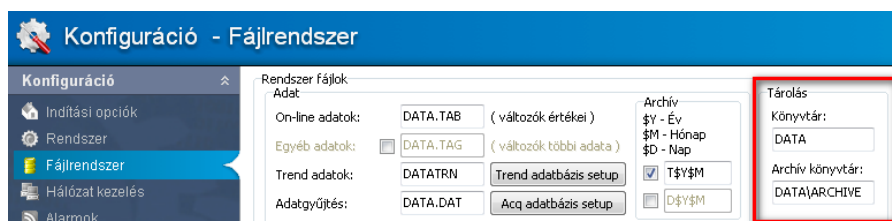
Időről időre felvetődik a program telepítési könyvtárának írásvédelme, hogy abban se szándékosan, se véletlenül ne lehessen módosításokat eszközölni. Igazából ennek eddig sem volt különösebb akadálya, hiszen egy bekonfigurált VISION alkalmazásnak nem szükséges többé írnia még a VPR.INI fájlt sem. Ezekre a program nem is jelez és eddig sem jelzett hibát. **A rendszer szempontjából csak egy olyan könyvtár létezik, amelyiknek írhatónak kell lennie: a ...Tmp alkönyvtár.** Mivel itt tárolja a rendszer a működése során keletkező összes átmeneti fájlt – például nyomtatáskor, vagy SVG konvertálás során.

Ez azonban mostantól átirányítható:



A fenti példán a *<Temporary>* megadás a mindenkor felhasználó *AppData* lokális átmeneti könyvtárát jelenti, de beírható ide bármilyen más, létező könyvtár is, amit a program az *Alkalmaz* gomb megnyomásakor ellenőriz. Innen kezdve az egész telepítési könyvtár írásvédetté tehető.

Van azonban még egy probléma: az alkalmazás, ami gyakran a telepítés alatt a *Project*-ben található. Azonban ennek az írhatóságára sincs szükség, ha maga az alkalmazás nem ír abban semmilyen fájlt. Hagyományosan a *Data*, a *Misc* és az *Alarm* alkönyvtárakat érinti ez a megfontolás. Miután a legtöbb alkalmazás ma már SQL-ben kezel csak adatokat, a *Misc* és az *Alarm* alkönyvtárak szerepe is megszűnt, a *Data* pedig átirányítható – hasonlóan a Tmp könyvtárhoz:



Így végül egyetlen olyan rendszer szinten kezelt könyvtár sem maradt, ami írhatóságot igényelne. Már csak az szükséges, hogy a rendszerbe épített írhatóság ellenőrzést kikapcsoljuk (mivel alapban ez működik). Pontosán erre szolgál az első képen a checkbox az átmeneti könyvtár mellett: *alkalmazás könyvtárának írás / olvasás ellenőrzése*.

Arra azonban felhívnom a figyelmet, hogy az automatikus update funkció innen kezdve nem működik!

X10 Verzió 2.2

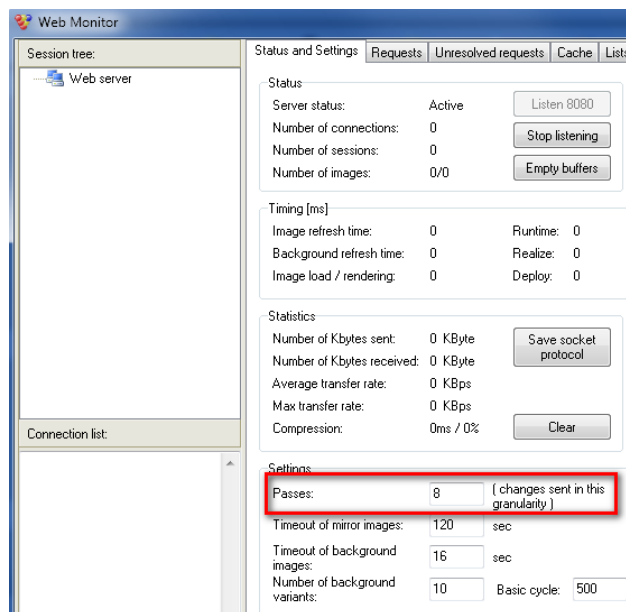
1. Web ciklusok (2.2)

A kép- és web ciklusok összerendelésében történt néhány apró változtatás, nevezetesen még progresszívebb lett az összerendelés. A gyorsabb képek a web-en is gyorsabban ketyegnek, a lassabbak pedig még lassabban. A módosítást az indokolta, hogy nagyobb számú klienst is ki tudjunk szolgálni abban az esetben, ha a gyors képfrissítés nem indokolt, és az esetek egy jelentős részében ez a helyzet – sokszor még a kvázi statikus kép is elegendő, ami a kép felrajzolásakor érvényes állapotot mutatja és kimerevíti. Konkrétan 5000ms vagy a feletti ciklus esetén ez a helyzet.

Az összerendelés a következő:

Kép ciklusidő	Web ciklus
Cycle = 10-500 ms	fix 500 ms
Cycle = 501-999 ms	fix 1000 ms
Cycle = 1000-1999 ms	fix 2000 ms
Cycle = 2000-4999 ms	fix 5000 ms
Cycle >= 5000	fix 0

A teljes ciklus persze függ még a lépésszámtól, amit itt tudunk konfigurálni és alapértelmezett értéke 8:



Vagyis pl. 500 ms képciklus esetén, amikor a Web ciklus is még 500 ms, a teljes késleltetés valójában 4 másodperc lesz ($8 * 500\text{ms}$), de a lépésszám pont ugyanaz, mint a desktop képen.

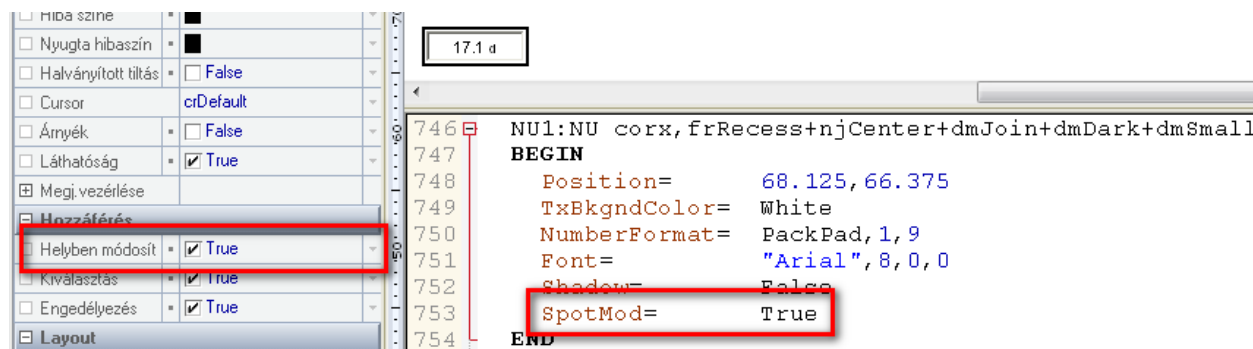
Ez a valóságban azt eredményezi, hogy bár pont ugyanazt látjuk és pont ugyanolyan gyakran változni a web-en, mint vastag kilensen, 4 másodperccel képsőbb. Ez nem új tulajdonság (csak a ciklusidők hozzárendelése az), és azért van rá szükség, hogy a tranzakciók számát csökkentsük a szerver felé. *Passes=1* beállítással ezt kikapcsol-hatjuk, vagyis elérhetjük, hogy késleltetés egyáltalán ne legyen. Csak akkor a szerver fél másodpercenként fog kapni a tallózótól egy

frissítés kérést!

Ezeket tudva kell az optimális beállítást elvégezni. Sok alkalmazás tapasztalata alapján a képciklus 500 ms-ra állítását javaslom (ez bőven elég a legtöbb ipari folyamat online megjelenítésére), a lépésszámot (Passes) pedig a default 8-ra. Más, főleg inkább adatbázist kezelő alkalmazásoknál a ciklikus frissítésnek nincs értelme, ezért itt a lépésszám 1-re, a ciklusidő pedig 5000ms-ra állítható. Ilyen az Energrade program is például. Ez utóbbi beállítás azt eredményezi, hogy az alkalmazásunk pont úgy működjön, mint bármilyen adatbázis kezelő alkalmazás a Web-en: csak akkor megy kérés a szerver felé, ha a felhasználó valamit állít, kiválaszt vagy módosít. Ilyenkor a 64-bites VISION akár 500 egyidejű felhasználót is képes kiszolgálni. Míg a ciklikus frissítést is igénylő rendszerekben csak kb. 50-et.

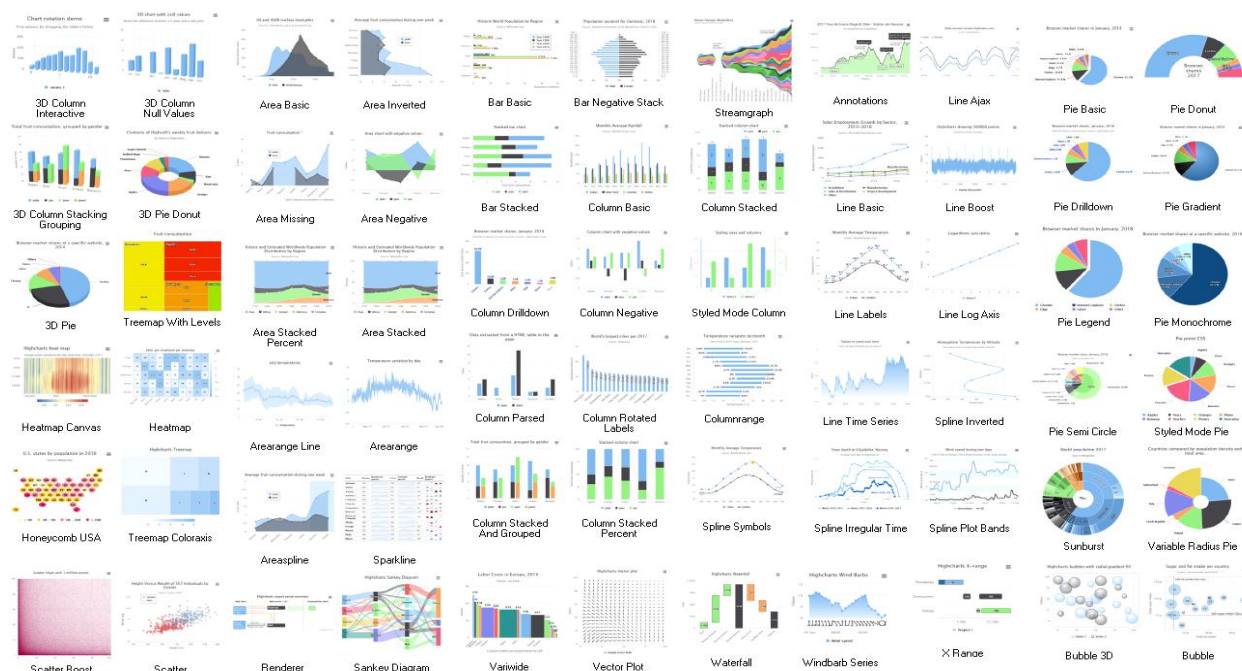
2. Feltételes adatmódosítás Web-en (2.2)

A Web-en nem lehet jobb egérgombbal a kijelzett értékre kattintani, csak bal egérgombbal. Ilyenkor is lehetséges azonban az adatmódosítás, amennyiben a *Helyben módosít* (*SpotMod*) tulajdonságot bekapcsoljuk. Ettől kezdve a Web-en is hasonló módon viselkedik majd a program: rákattintunk egyszer, mire az illető számkijelző szerkeszthetővé válik – vagy Up-Down box formájában, ha analóg, vagy legördülő listaként, ha diszkrét vagy többállapotú – és automatikusan visszavált 6 másodperc múlva. Közben lehetséges az adatmódosítás. (Természetesen az IN objektum továbbra is működik.)



3. Highcharts

A világ vezető chart rajzoló programja mostantól elérhető a VISION-ben is. Segítségével több, mint 100 féle chart közül választhatunk:



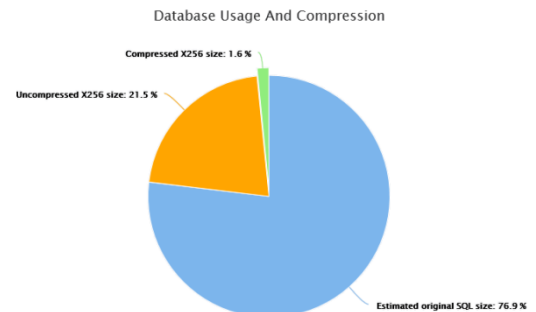
A chart maga a CreateChart függvénnyel készül, ahogy ezt a már a trend SQL diagnosztikánál is láthattuk. A részletek azonban szétfejtik ennek az összefoglalónak a kereteit, ezért érdeklődés esetén tanfolyami témának ajánljuk.

F fontos tudnivaló: a Highcharts külön license megvásárlását igényli!

3. Big Data (2.2)

A trend SQL Big Data mellett megjelent egy új Big Data technológia is a VISION-ben, amivel még nagyobb mennyiségű adat kezelésére nyílik lehetőség, ráadásul az eddigieknél is sokkal gyorsabban. A program új demo-ja már ezt a fájlformátumot használja az energiamenedzsment demón belül. Egy pilot projektben az alábbi adatstatistikát mértük:

Mérőváltozók száma:	6 616 721
Teljes adatmennyiség:	393 737 490 090
Teljes (tömörítetlen) adatbázis méret:	886 371 554 810
Tömörített méret:	52 455 549 306
Tömörítés mértéke:	1,6%
Tárolási hatékonyság:	0,133 byte/adat

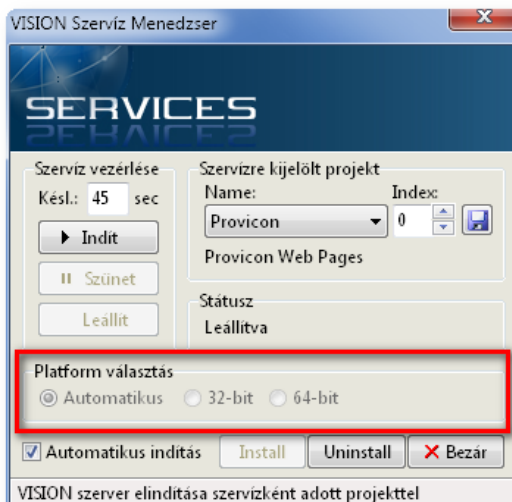


A charton jobb oldalt az látszik, hogy az X256 adatbázis mérete mindössze 1,6%-a az eredeti feltételezett MS-SQL méretnek. A tömörítés mértékére jellemző, hogy **1 adat elfér 1 biten!**

4. Platform választás szervízben (2.2)

A VISION szervíz menedzserének új változata lehetővé teszi, hogy attól függetlenül válasszunk 32- vagy 64-bites szervízek között, hogy egyébként az operációs rendszer milyen. Ezzel lehetővé válik, hogy 32-bites szervízként futtassuk a VISION-t 64-bites környezetben is (néha a kommunikációs driver miatt lehet rá szükség, ha a kérdéses driver 64-biten még nem lenne elérhető). Ez a választás a korábbi változatban nem volt lehetséges.

Az új beállítás látszik alább:



A *Platform választás* menüpont az új – itt lehet meghatározni, hogy továbbra is automatikusan válasszunk platformot, vagy magunk határozzuk meg azt. A program a kiválasztott opciót megjegyzi.

Fontos hangsúlyozni, hogy **a beállítás csak a komponens installálása előtt lehetséges**, hiszen a Windows szervíz menedzserébe a teljes elérési út is rögzítés kerül, és ugye a 32- és 64-bites szervízeknek más az elérési útja: 64\.

Nem új funkció az *index*, de nem árt emlékeztetni rá: ez egy újabb opció installálás előtt, hogy egy rendszerben több *VisionServer*-t is tudjunk üzemeltetni. Az installált *VisionServer* neve mögé az *index* beíródik, így egy rendszeren belül akár 10

Vision szervert is installálhatunk különféle projektekkel és akár eltérő platformon. Így akár kombinálhatjuk is a 32- és 64-bites szervízeket egyazon rendszeren belül, ha arra szükség lenne.

A 9.64.19+ változatokban ezenkívül történt néhány apróbb hibajavítás – a részletek a honlapon megtalálhatók. Nem csak az új szervíz menedzser miatt érdemes tehát telepíteni az új rendszert, de a szervíz szolgáltatást magát is érdemes upgrade-elni.

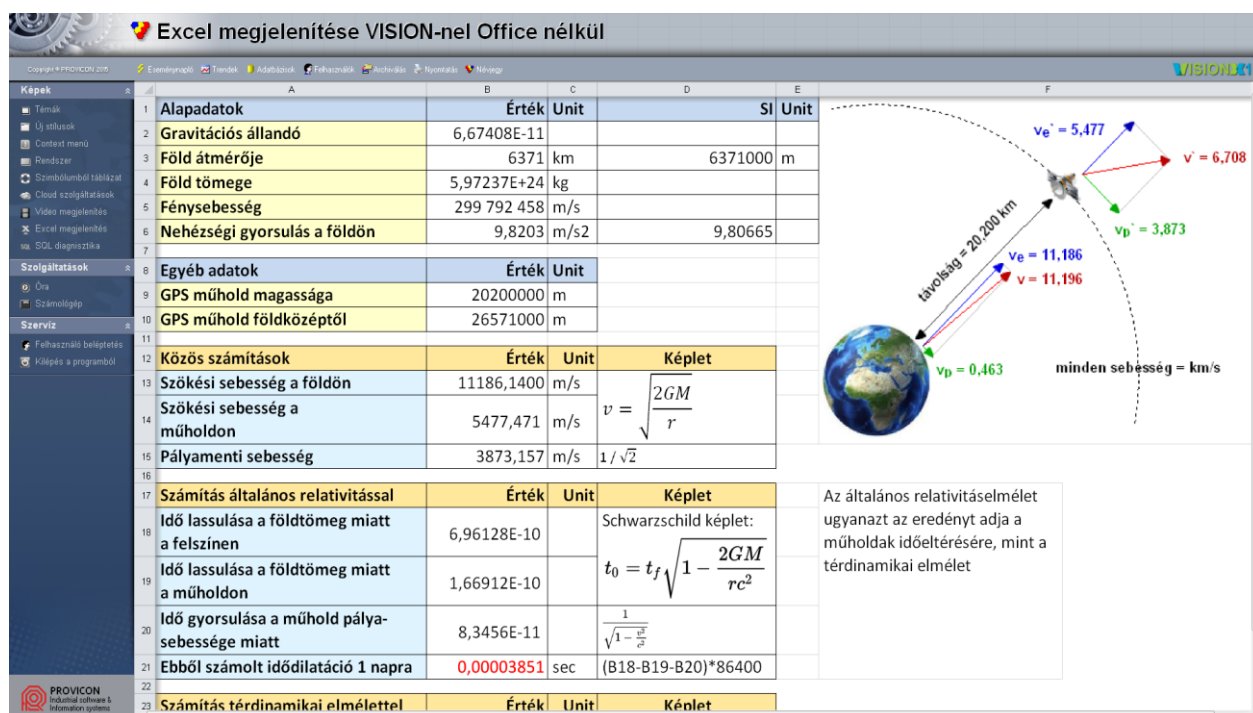
5. Excel megjelenítése Office nélkül (2.2)

Az embedded (beépülő) Excel funkcionalitás a VISION-ben további funkcióval bővül: Excel munkalapok megjelenítése. Maga a funkció az AdvancedCourse demo alatt tanulmányozható, onnan

a megfelelő XWND objekt (ez új) kimásolható. Az XWND objektum persze jóval többet tud ennél, valójában egy mini X10 keretrendszer, de ennek részleteiről külön dokumentumban fogunk írni. Mindeenesetre az Excel megjelenítésére elég könnyű használni, ezt kell mindössze beírni az alkalmazásba:

```
XWND1:XWND WindowWidth-24.5,WindowHeight-17.5
BEGIN
  Position=      23.875,0.125
  Content=       wcGrid
  Parts=         0
  MultiCell=     false
  GridSpec=      "Excel\SAT.xlsx"
  Anchor=        akLeft+akTop+akRight+akBottom
END
```

A megjelenített Excel fájl teljes egészében megfelel az Office változatnak, habár ebben az esetben az összes cellát, szöveget, képletet, bitmapet stb. a VISION értékeli ki és rajzolja – az Excel-nél sokkal gyorsabban – a képernyőre:



Ez a példa önmagában is bonyolult képleteket, cella formattálást és PNG képet is tartalmaz.

6. SVG fájlok megjelenítése korlátozások nélkül (2.2)

Az SVG egy ideje elérhető már a VISION-ben, de eddig csak az 1.0 változata működött, ami a transzformációk tekintetében elmaradt a teljes megoldástól. Mostanáig. Az új VISION (19+) változattól azonban ez a korlát is megszűnt, a rendszer automatikusan váltogat 1.1 és 1.0 SVG fájlok között. Ráadásul a megjelenítésük a bonyolultságuk ellenére rendkívül gyors, mivel a program belül PNG-vé konvertálja őket. Íme, egy példa:



X10 Verzió 2.1

1. Jelentős SQL sebesség növekedés (2.1)

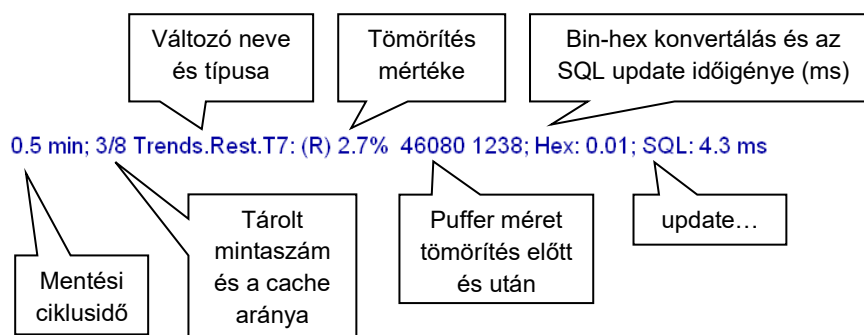
Egy új SQL ADO komponensnek köszönhetően az új verzióban majdnem 100-szorosára(!) gyorsult fel a nagy eredményhalmazok kiolvasási ideje. A módosítás a végrehajtás típusú műveletekre is hat (*insert...update.. stb.*), de rájuk csak kisebb mértékben, a sebesség növekedés elsősorban a lekérdezéseknél (*select...*) látványos. A módosítás érthető a nagyméretű alkalmazói adatbázisok megjelenítését táblázatban, trenden, charton, a rendszer adatbázisai közül pedig elsősorban az esemény naplót.

2. SQL alapú trend és adatgyűjtés monitorozása és diagnosztikája (2.1)

Két új függvénnyel kiirathatjuk egy szervíz képre az SQL adatbázis műveletek státuszát, sebességét, a pufferek méreteit menet közben. Prototípusai:

```
SQLTrendProgress:: namestring function 'returns a string about SQL trend progressing
SQLAcqProgress::   namestring function 'returns a string about SQL data acquisition progressing
```

Legegyszerűbben egy TX objektum szövegparaméterének adhatjuk át a fenti függvényt egy erre kijelölt szervízképen, az eredmény pedig, amit a képen látunk, egy folyton változó string a mintavételezés és tárolás ütemében:



A mentés ciklusideje a tárolási ciklus végén számíttódik ki és azt mondja meg, hogy mennyi minta van a memóriában, vagyis a cache-ben. Abnormális kilépés, kikapcsolás esetén ennyi adatot veszíthetünk el.

A cache tárolási kapacitásának és kitöltöttségének az aránya optimális esetben 50% körüli – a rendszer induláskor így számolja ki, így képez 100%-os tartalékot a memóriában arra az esetre, ha a ciklusok nem tudnának pontos ütemezés szerint lefutni. Emlékeztetnék, hogy a *Trend save..* üzenet utal arra, amikor a cache már a mintavételezés során megtelik, azaz a trend mentési ciklusok lemaradnak a minta-vételezéstől. Fontos azonban tudni, hogy adatvesztés ebben az esetben sincs, mivel az adatokat ilyenkor a program a mintavétellel egyidőben menti.

A rendszer szinte csak *update* utasításokat ad ki az SQL szerver felé, kiolvasás (*select*) menet közben nincs, csak naponta egyszer, napváltáskor. Az SQL update időigényét láthatjuk fent – általában 1-10 ms között változik. Nagy rendszerekben másodpercenként akár 10 ilyen műveletre is sor kerülhet, ami még 10ms esetén is alig terhel (10%). A konkrét ciklusidő kiszámításánál a rendszer megpróbálja a cache-ben tárolt adatmennyiséget 1 órára korlátozni.

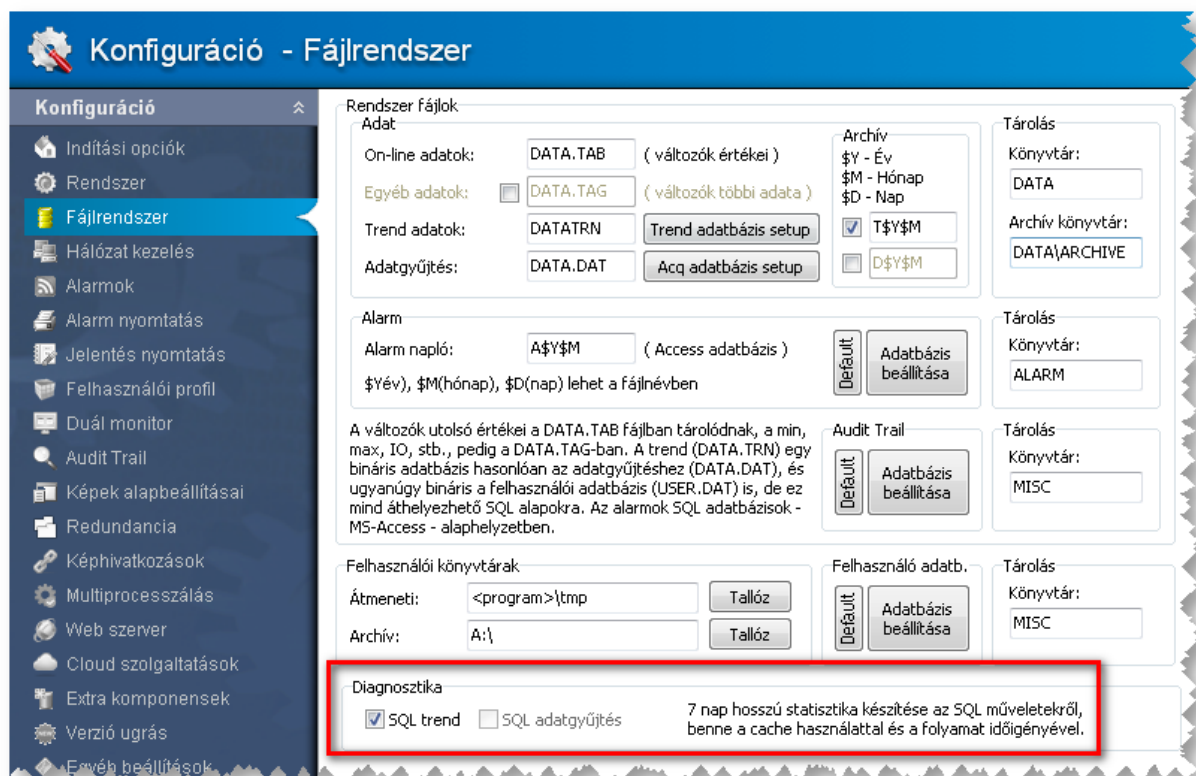
A fenti két függvény elsősorban diagnosztikai célra használható, mély elemzések elvégzését támogatja. Feltérképezhetjük, mennyire terheli a rendszerünket a trend mintavételezés és a cache

tartalmának az SQL adatbázisba mentése, ill. hogy a cache méret elegendő-e a minták kényelmes, időben elosztott tárolására.

A fenti real-time adatokból a program statisztikát is képes készíteni, amiben az adatok 5 perces átlagát tárolja el 7 napra visszamenően. A statisztikába tehát minden változó beleszámít, vagyis az átlag cache kihasználás jelenik meg benne, valamint az 5 perces SQL trend adatbázis műveletvégrehajtási idő százalékosan. Ez utóbbi adja a valódi SQL terhelést, hiszen perces léptékben már igen nagy pontosságú adataink vannak. Nagyobb rendszerekben kb. 1000 miliszekundum pontosságú adatból számítjuk ki a terhelést.

Az SQL terhelésnek illik 10% alatt, a cache kihasználtságnak pedig 50% körül lennie. Ezen felül a program kiszámítja a tömörítés átlagos mértékét is, ami azonban már nagymértékben rendszer, ill. adat függő.

A diagnosztika és statisztika készítés engedélyezése a konfigurációs ablakban lehetséges:



Ennek hatására a program az alkalmazás könyvtára alatt egy *Diag* alkönyvtárat készít és abba egy *json* kiterjesztésű fájlt tárol. Ez a fájl használható a *CreateChart* függvény argumentumában, amivel a statisztikát egy diagnosztikai képen megjeleníthetjük:

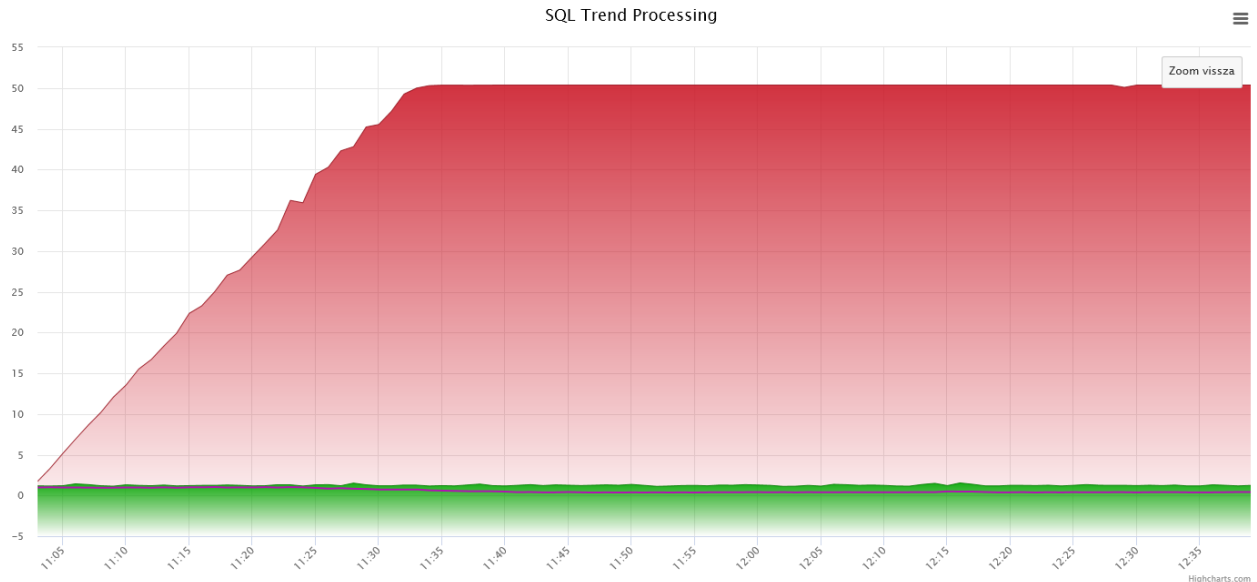
```
IB1:IB WindowWidth-25.25,85.625,CreateChart("line-time-series","trend","Diag\TrendStat.json",19,"SQL Trend Processing")
BEGIN
  Position= 25.75,WindowHeight-105
  Anchor= akLeft+akTop+akRight
END
```

Hagyományos **IB-t** (Internet Browser) használunk tehát, aminek argumentuma a *CreateChart()* függvény kimenete. Ez a függvény végzi el a konkrét chart-script előállítását és abba *TrendStat.json* kód beillesztését. A *CreateChart* paraméterei a következők:

- Template neve ("line-time-series")
- Chart típusa ("trend")

- Json adatfájl
- Chart formátuma
- Chart címe (opcionális)

Ennek hatására az alábbihoz hasonló trendképet kapunk:



Ezt a képet érdemes tehát egy diagnosztikai képre kitenni. A chart formattálás lehetséges értékei a következő bitek kombinációja (4-ik paraméter, integer érték):

- 1: Vízszintes rács
- 2: Függőleges rács
- 4: Mérnöki mértékegység a rácson (%)
- 8: Szaggatott rácsvonal stílus
- 16: X skála forgatási szöge (0: vízszintes, **16: 315 fok**, 32: 270 fok, 48: 90 fok)
- 64: Keresztező vonalak pásztázáskor
- 128: Kétoldalú skála

A példában megadott 19-es érték (1+2+16) mindkét irányú rácsot adja meg és az X skála feliratait 315 fokban elforgatja a képen látható módon.

Íme a példában szereplő parancs, hogy könnyebb legyen a CreateChart függvényt és annak argumentumait kimásolni a dokumentációból:

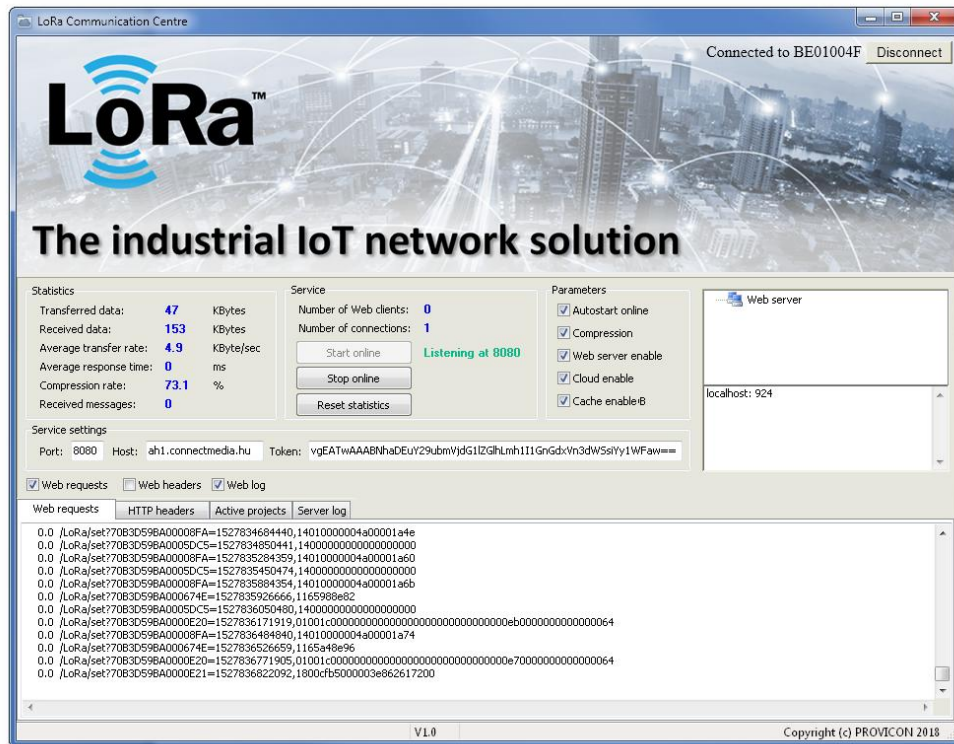
```
CreateChart("line-time-series","trend","Diag\TrendStat.json",19,"SQL Trend Processing")
```

X10 Verzió 1.4

1. LoRa kommunikációs driver (1.4)

Az SB-Controls Kft.-vel együttműködésben elkészült a LoRa kommunikációs driver a VISION-höz. Egy univerzális, kombinált megoldás született, ami egyszerre támogatja a szolgáltatónál (pl. Antenna Hungária Zrt.) regisztrált eszközök leolvasását, valamint a saját LoRa szerverrel, lokális hálózatban (tehát saját hatáskörben) működtetett eszközök kezelését (kerlink).

A megoldás alapja egy önálló LoRa kommunikációs szerver, aminek a képe alább látható:



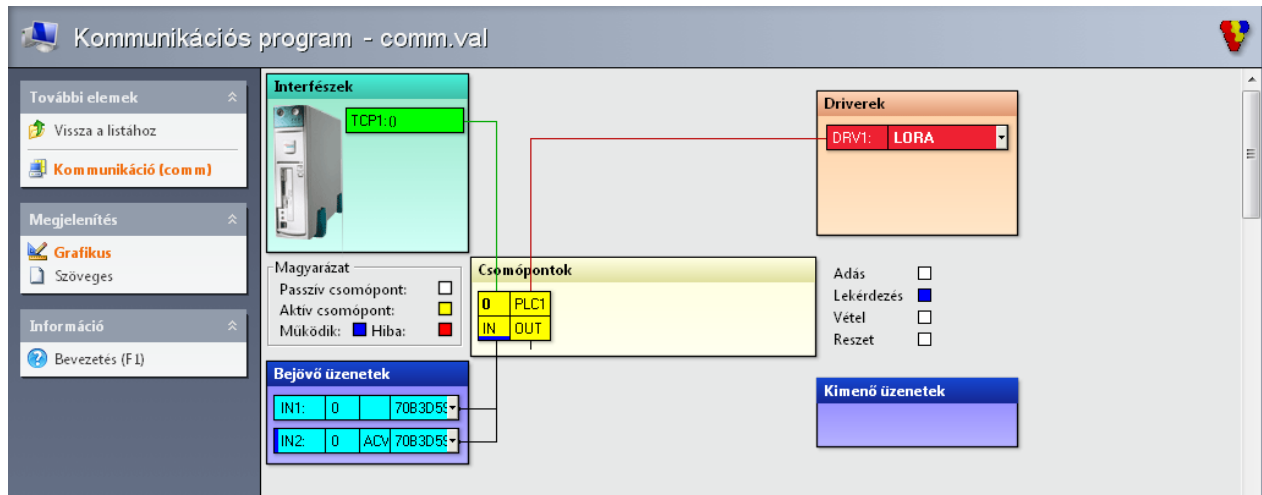
A szerver 1000 eszköz kezelésére képes. A szerver képen alul, a Web request monitoron látható az eszközök *Payload* listája, ahogy a szolgáltatótól leolvassuk az adatokat. Abban az esetben pedig, ha saját szerverből olvassuk azokat, akkor csak egy `/Lora/rx_data` parancs látható. Ezt az üzenetet a tesztkörnyezetben egy SB-Controls-tól beszerezhető *kerlink* nevű eszköz produkálta, amit a következőképp kell beállítani:

Adatok továbbítása:

Protokoll: HTTP
 Port: ahogy konfiguráltuk (pl. 8080)
 Path: /LoRa (ezt egészíti ki a kerlink a `/rx_data` paranccsal automatikusan)

Természetesen a tűzfalon engedélyezni kell, hogy a LoRa szerver a megadott porton fogadni tudja az adatokat (inbound rule).

Maga a szerver a VISION telepítési könyvtára alatt, a Comm-ban található és LoRaSRV névre hallgat. Ezt azonban nem kell manuálisan elindítani, mivel azt a LoRa nevű kommunikációs driver elintézi. A használathoz elég tehát bekonfigurálni a LoRa drivert, majd ahhoz 1db default csomópontot csatolni és hozzá tetszés szerinti számú Input üzenet blokkot:



Mindezt tetszés szerinti számú taszkban elkövethetjük, a LoRa driver valójában nem kommunikál, csak a szervert. A driver a szerverből olvassa át közvetlenül az adatokat.

Alaphelyzetben az üzenet blokk a byte sorozattá alakított ún. *Payload* adatsort tartalmazza, annak feldolgozásáról alapesetben nekünk kell gondoskodni. Azonban a LoRa eszközök egy részéhez készítettünk egy dekódert is, amit egyszerűen adattípusként választhatunk ki az üzeneteknél:

Azonosítás	
☐ Név	IN2
Hozzárendelés	
☐ Üzenet száma	0
☐ Adattípus	ACW-TH
☐ Cím	70B3D59BA0006791
☐ Többállapotú bitek	mbInput
☐ Timeout	
☐ Frissítési idő	0

Az adattípus a listából kiválasztható. Alatta a cím mezőben látható az eszköz 16 karakter hosszú saját azonosítója.

Például az ACW-TH hőmérséklet és páratartalom mérő esetén a definíciós sor a következőképp alakult:

ACW-TH|R32=[2,2]/65536*175.72-46.85|R32=[4,2]/65536*125-6

Ebben az esetben a device leíró három mezőből áll, az egyes mezők | oszlop karakterrel elválasztva. Az első mező tartalmazza a device nevét (ezt listázza az adattípus sora). Az ezt követő mezők pedig az adatokat definiálják az üzenetben. A [2,2] rész ezen belül azt jelenti, hogy az eredeti Payload adatsor 2-ik byte-jától kezdve olvasunk 2 byte-ot. Az eredmény pedig az ott olvasható képlet szerint számítható, amit a gyártótól kaptunk (ld. jobb oldalt). Ezáltal rendkívül kényelmessé válik az adatok dekódolása, hisz csak egyszer kell megfejtetni, hogy melyik adat, hol található, azután beírni a definíciós fájlba – adott esetben egy képlettel továbbszámolva.

$$T(^{\circ}\text{C}) = \frac{\text{TemperatureCode} \times 175.72}{65536} - 46.85$$

$$H(\%RH) = \frac{\text{HumidityCode} \times 125}{65536} - 6$$

A device leíró definíciós fájl neve *LoRa.def* és ugyanott, a Comm alatt található. A képlet elején látható R32= rész azt határozza meg, hogy az eredményt milyen formában szeretnénk átadni a VISION-nek. Az R32 például azt jelenti, hogy realként 4 bájton. Az alábbi típusok léteznek:

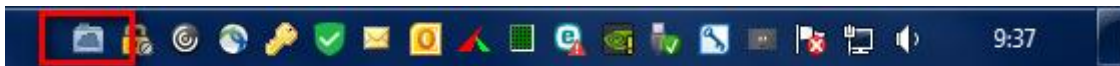
Adattípus	Típus	Hossz
R32	Real	4 byte
R64	Double	8 byte
I8	Integer	1 byte
I16	Integer	2 byte
I32	Integer	4 byte

Az adatsor eleje a táviratban mindenképp tartalmazza az adathoz tartozó időbélyeget is packtime (tehát VISION-ös) formátumban. Ez azért szükséges, mert az adat nem real-time, általában 15-perctől 1-2 óra a késleltetése. Adott esetben nem árt tehát megjeleníteni az időt is, amihez az adat tartozik.

Másfelől az időbélyeg az adatok utólagos visszatárolását is elősegítheti, ui. a szolgáltató, de a kerlink is tárolja a még le nem olvasott adatokat, amit akár több nap múltán is kiolvashatunk, az időbélyeg pedig mindig az épp feladott adathoz tartozik. Az adatokat így visszaírhatjuk akár a trendben vagy saját adatbázisunkban – időhelyesen.

Rendszer használata

A LoRa kommunikációs rendszer használata a ModTCP slave szerverhez hasonlít. Ott is külön szerver készült a közös, nagyszámú eszközt kiszolgáló kommunikációra. Ahogy abban az esetben is, a LoRa szerver is elindítható kézzel, a VISION indítása előtt. Ebben az esetben a VISION nem fogja külön elindítani és a VISION bezárása sem fogja leállítani azt! Még egy fontos különbség van a VISION által indított (és automatikusan leállított) szerver, valamint a kézzel indított szerver között. Az előbbi esetben a szerver processzként, rejtett módban fut, amiről csak a tálcán látható kis ikon árulkodik:

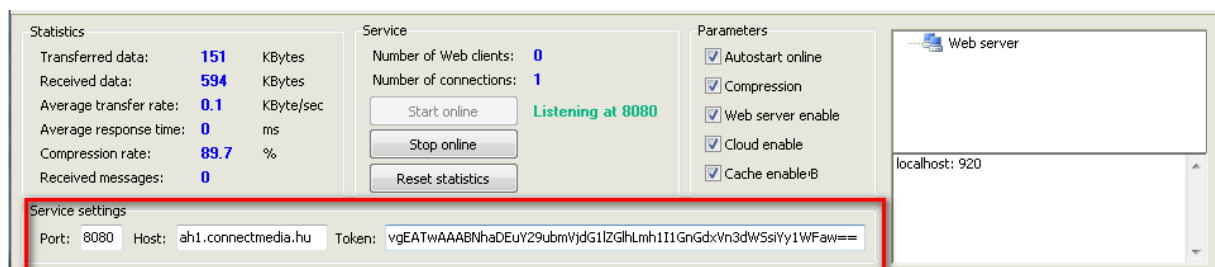


Erre kattintva azonban meg is jeleníthetjük a fejezet elején bemutatott szerveret. Ha ellenben kézzel indítjuk, akkor alkalmazásként fut és mi is állíthatjuk le szükség szerint.

Miért fontos azonban mindez? Elsősorban azért, mert a LoRa kommunikációs rendszerek eltérnek kissé a real-time kommunikációtól, új adat viszonylag ritkán (pl. óránként) keletkezik. Viszont mindaddig, amíg nincs friss adat, a VISION üzenetei is timeout-ra futnak. Ha ellenben a szervert már korábban elindítottuk, akkor a VISION az újraindítása után azonnal átveszi onnan a legfrissebb adatokat.

LoRa szerver

A szervert mindenek előtt be kell konfigurálnunk *Port*, *Host* és *Token* tekintetében. Ez utóbbi adatot kapjuk a szolgáltatótól, miután regisztráltunk.



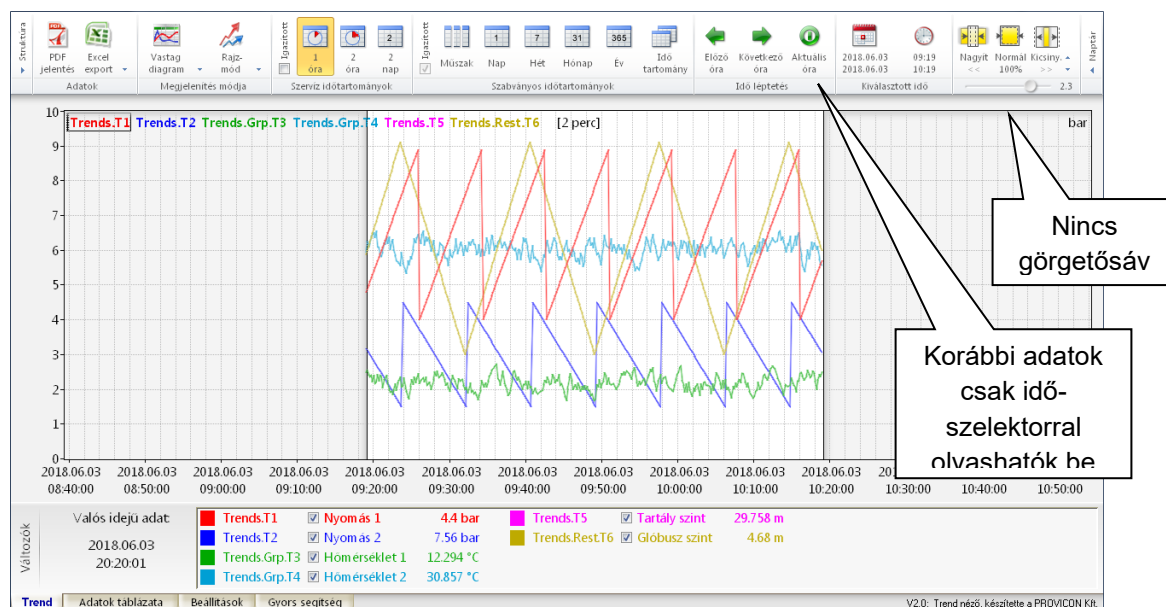
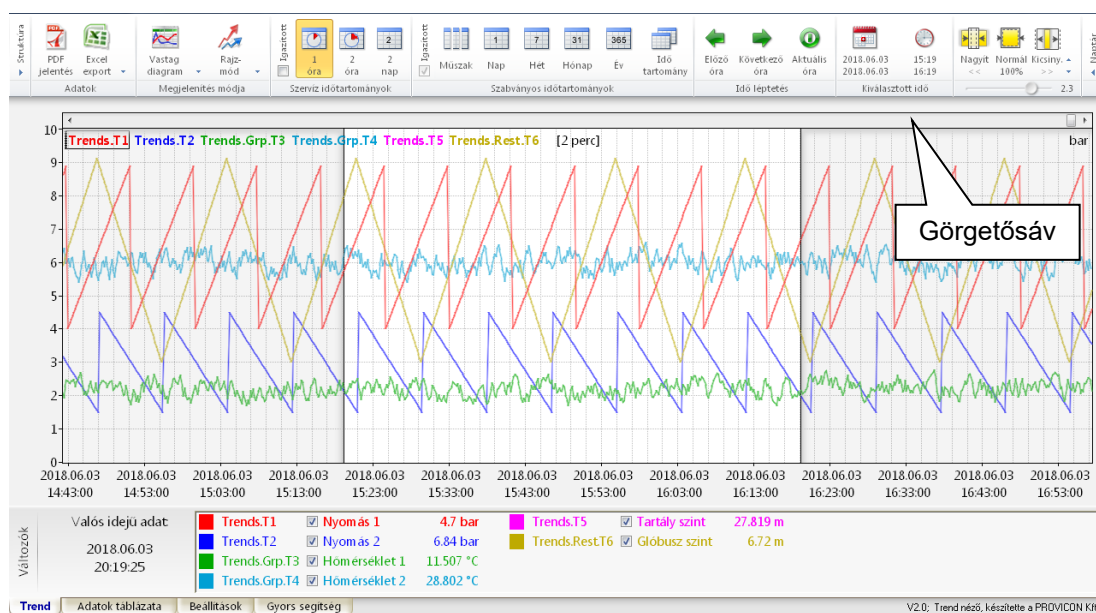
Mire szolgál azonban a *port*? Nos, a LoRa szerver egy önálló Cloud szerver is egyben, ami más Cloud alapú rendszerek felé önmaga is szolgáltat adatokat. Mégpedig úgy, ahogy ezen a tallózóból kivágottnak láthatjuk (megadjuk a szerver IP címét és portját, majd utána *getpos?device*:



2. Direkt trend-adatbázis ablak (1.4)

Egy új trend tulajdonságnak (*TimeFit*) köszönhetően lehetővé vált tetszőleges hosszúságú SQL trend adatbázis visszaolvasása és megjelenítése. A megoldás lényege, hogy kikapcsoljuk a trend görgetősávját és a trendbe kizárólag az adott időszelekciónak megfelelő adatokat olvassuk. Pontosan úgy, ahogy direkt adatbázis tábla hozzárendelés esetén. A trendbufferben ilyenkor mindig annyi adat van csak, amennyi a képernyőn látszik és mindegy, hogy milyen régi adatot olvastunk bele.

Hogy jobban érthető legyen, nézzük meg az alábbi két trend ábrát:

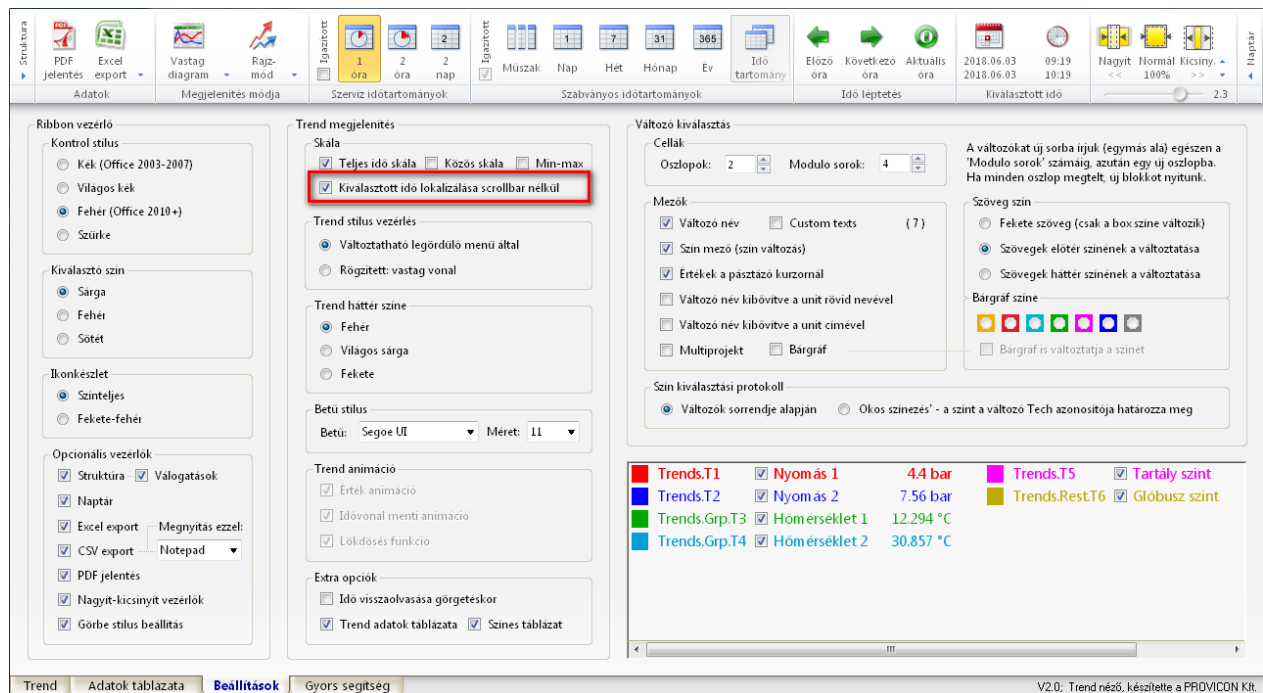


Felül a hagyományos, alul az új opciónak megfelelő trend látható belenagyítás után (hogy láthatóvá váljanak a szomszédos mintaelemek). Mint ismeretes, a fehérrel kiemelt rész felel meg a kiválasztott időnek, előtte-utána a buffer további tartalma látható. Azonban csak a hagyományos esetben, mivel a trendet egyébként a teljes hossz mentén görgethetjük a maximális mintaszámig. Ez pedig korlátozott. Mert hiába a 64-bit, 1 másodperces mintavétel mellett dőreség lenne mondjuk 5 év adatát egyben beolvasni és teljes hosszában görgetni. Egyrészt mert nagyon nagy memóriát igényelne, másrészt

eléggyé lassú lenne az összes adat beolvasása miatt. Ráadásul görgetni se nagyon lehetne, mert 1 pixel görgetésre az adat napokat ugrana.

Ezért szerencsésebb ilyenkor a második módszer használata, ami csak az időszelekciónak megfelelő adatokat olvassa be. Ez jóval gyorsabb, nincs időbeli korlátozás és kisebb memóriát is igényel.

Egyebekben ugyanúgy nagyíthatjuk és mozgathatjuk jobbra-balra az így beolvasott adatokat, vagyis az új opció is pont ugyanúgy használható, mint a korábbi – csak a görgetősávról mondtunk le. Beállítása a *TrendWindow* konfigurációs oldalán található új opcióval lehetséges:

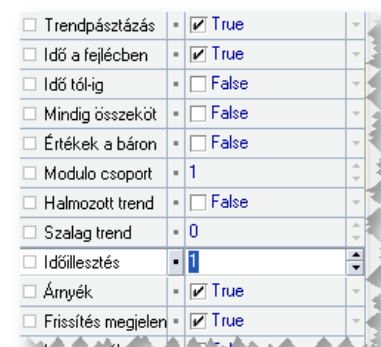


Opció: *Kiválasztott idő lokalizálása scrollbar nélkül.*

Végeredményben ez az opció egy direkt trend-adatbázis ablakot nyit a trend SQL adatbázisa felé, mintha csak magunk rendeltük volna hozzá a TD(X) objektum *TableName* tulajdonságához a *.TrendData(i)* adatbázis táblát és a *FieldList*-hez a mező (változó) konkrét nevét. Még egy fontos tulajdonsága van azonban a *TimeFit* opciónak: **átlagolás**. Abban az esetben, ha az időtartomány több hónap, vagy év, átlagot képzünk a mintákból és azt jelenítjük meg a trendben. Például egy vagy több év esetén napi átlagokat képzünk. Ez a megoldás a feldolgozhatóságot javítja, miközben tovább gyorsít.

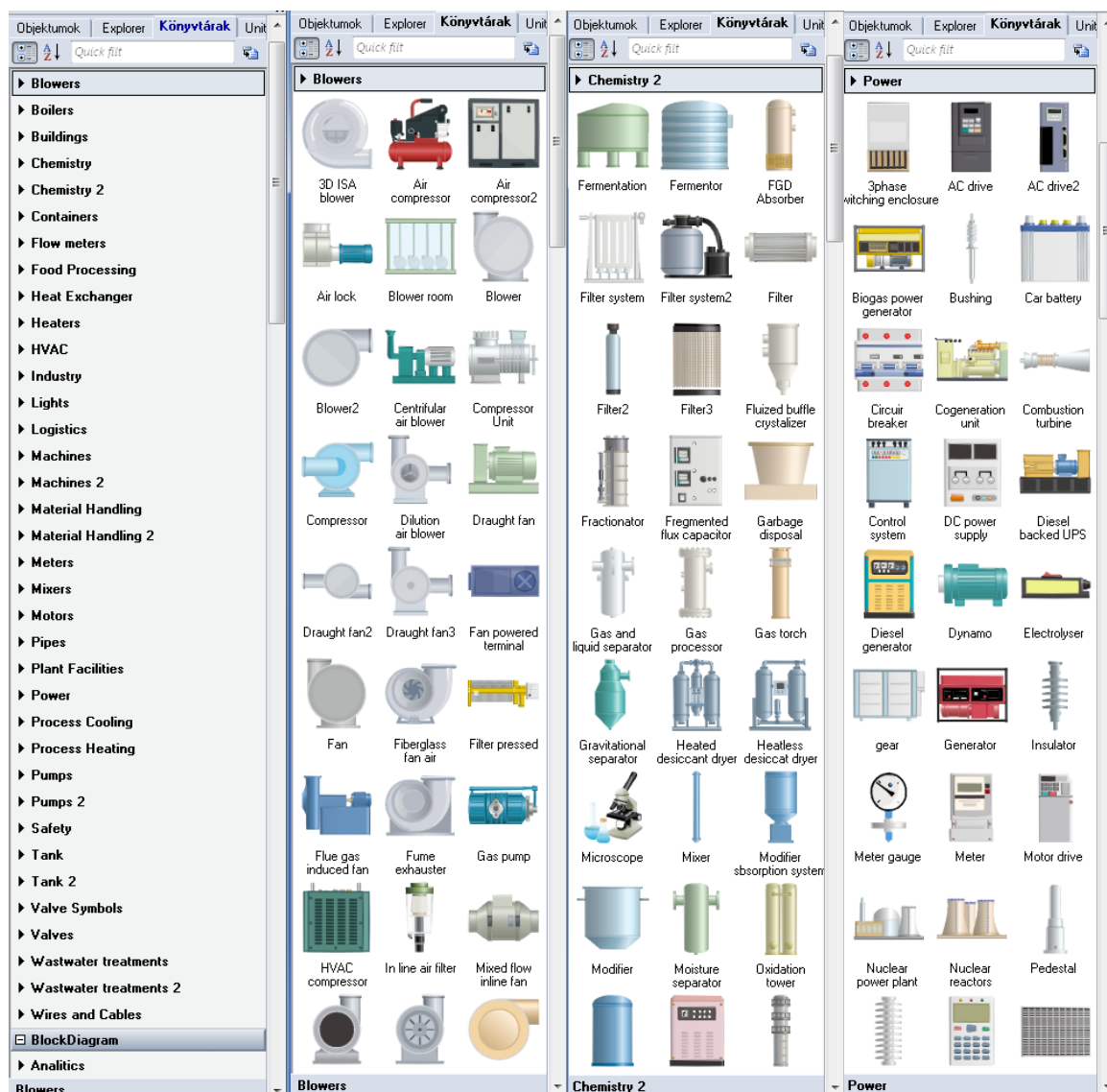
Saját képre rajzolt trendek esetén a *TimeFit* (*Időillesztés*) opciót kell 1-be írni az ábrán látható módon. **Azonban ez az opció csak időszelektorral együtt használható** (ez a lényege), vagy egy *TY* időszelektor objektumnak, vagy a *Globals* struktúrának a hozzárendelésével, ahogy a *TrendWindow* képen is csináljuk.

Az új opció Web-es környezetben is kényelmesebb, mivel ott a görgetősáv hossza – tekintve, hogy a Web-en az egész trendtartalmat egyetlen hatalmas PNG fájl formájában töltjük le – még inkább korlátozott. A *TimeFit* opció használata ezt a korlátot is megszünteti, miközben a letöltött adatmennyiség is csökken.



3. Több ezer új szimbólum! (1.4)

A 17-es verzió, 5000(!) új, vektoros VISION szimbólummal bővült, amit szerkesztés közben a könyvtárak fülön választhatunk ki:



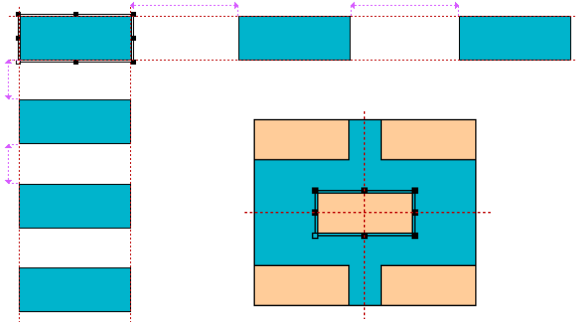
Az SY-vel felrajzol szimbólumokat tetszés szerint színezhethetjük és az alkalmazás könyvtárába is "lokalizálhatjuk", azonban ez utóbbit csak az objektum kiválasztó tetején található lokalizáló gomb megnyomása után. Egyébként a szimbólumra a könyvtárban hivatkozunk (hogy ne kelljen annyi szimbólumot bemásolni).

Ugyanitt található a szűrő, amivel kényelmesen leszűrhetjük a listát bármilyen keresett elemre (pl. valve).

X10 Verzió 1.3

1. Automatikus pozicionálás (1.3)

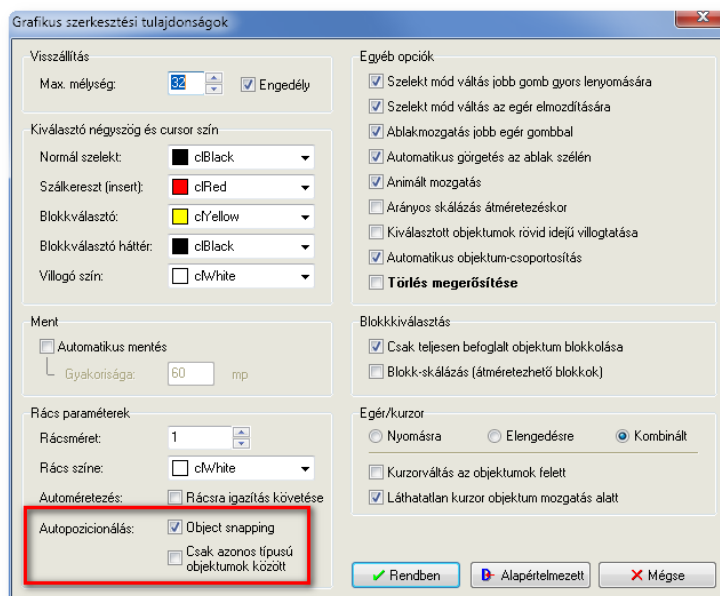
A grafikus szerkesztő egy praktikus funkcióval bővült: automatikus objektum pozicionálás (*object snap*). Az eljárásnak az a lényege, hogy mozgás közben a program automatikusan keresi azokat az objektumokat a képernyőn, aminek a széléhez vagy középhez hozzáigazítható az épp mozgatott vagy skálázott elem. Még érdekesebb, amikor az objektum elrendezését is rábizzuk erre a funkcióra:



A bal egérgombot megnyomva jelennek meg az ábrán látható pozicionáló vonalak a kiválasztott objektum környezetében és ilyenkor az alakzatot automatikusan rendezi is a program egyenlő vízszintes és függőleges térközök mentén.

A szélek bármelyikéhez hozzáigazítható a másik objektum bármelyik másik széle (a balhoz a jobb, a felsőhöz az alsó stb.). Hasonló megoldással a Microsoft Powerpoint-ban találkozhatunk.

Az ismertetett eljárás alpból része a 64.16-os verzióknak, de **kikapcsolható**. Különösen sok objektumot tartalmazó képeken lehet zavaró – pláne, ha a felhasználó nem szokott hozzá –, hogy vonalak sokasága jelenik meg a képen és az objektum a mozgatasakor automatikus pozíciókra gravitál. Kikapcsolásához a grafikus tulajdonságok beállító ablakát kell előhívunk (Ctrl+G), az **Object snapping** checkbox alul látható:



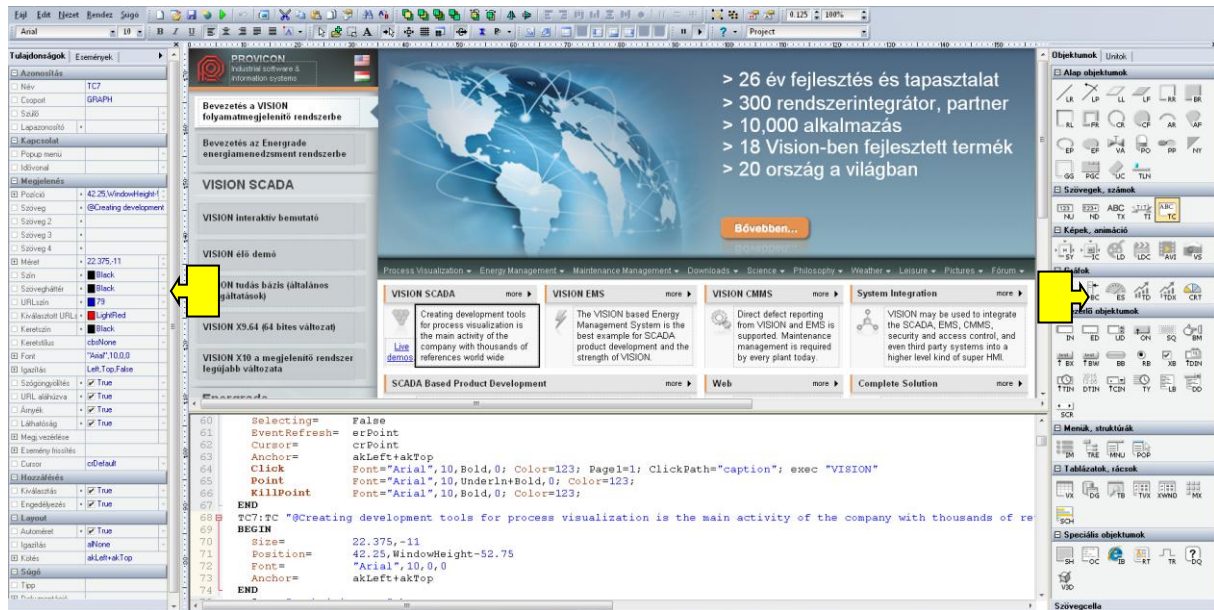
Összetett folyamatábrákon, ahol az objektumok nem feltétlenül illeszkednek más objektumokhoz, bekapcsolhatjuk a **Csak azonos típusú objektumok között** checkboxot, amivel azt érjük el, hogy szöveg csak szöveggel, szimbólum csak szimbólummal, bár csak bárral azono-sítsa a pozíciót.

A szerkesztő vonalak csak 1:1-es nagyítás mellett látszanak és nagyításban az automatikus pozicionálás sem működik – szándékosan. Mivel nagyítás-ban egyébként is kényelmesen tudjuk az objektumokat illeszteni, pozicionálni. Így adódott az ötlet, hogy az object snapping-et

ebben az üzemmódban kikapcsoljuk; így a felhasználónak sem kell a grafikus tulajdonságok beállító panelén ki-be kapcsolgatnia, elég ha simán ránagyít.

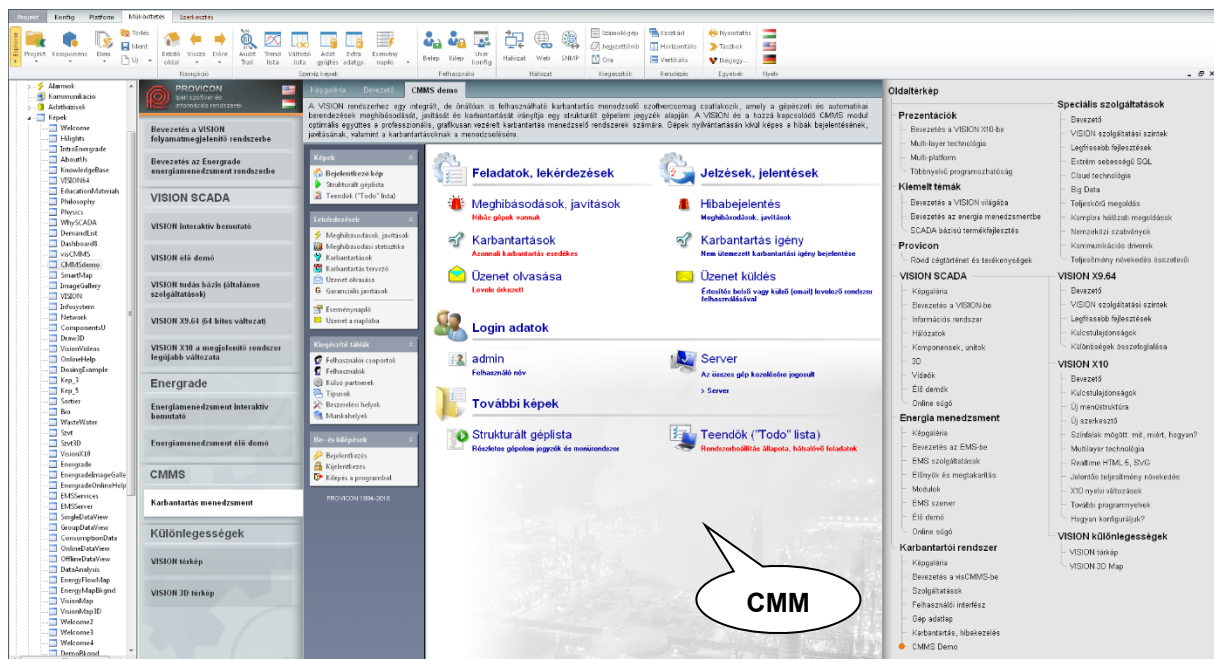
2. Szerkesztő ablak rekonstrukciója (1.3)

Az X10-es rendszerből átkerült pár praktikus technika az X9.64-be. Az egyik ilyen a szerkesztő ablakok elrendezése: a tulajdonságszerkesztő a szövegmező mellől a kép bal szélére került teljes magasságában, az objektum választó pedig ugyanilyen formában a jobb szélre. Ezáltal sokkal könnyebb a tulajdonságok között keresgélni, az objektum választó pedig önmagában képes megmutatni a szimbólumokat és a példa objektumokat (favorites):



3. OLE szervert funkció a CMMS programhoz (1.3)

Mostantól a CMMS programot akár az alkalmazás részeként is futtathatjuk, mintha a két program egy rendszer lenne. Nem kell tehát a két program között váltogatni a felhasználónak többé az Alt-TAB-bal vagy a tálcáról kiválasztani a futó programokat, mivel **a CMMS a VISION képbe ágyazva jelenik meg**, és mi döntjük el, hogy hol és milyen méretben:



A fenti kép az új demo programból lett kiollózva, a CMMS a kép közepén látható. A CMMS beágyazás módja ezen a képen tanulmányozható, ill. a saját alkalmazásba egyszerűen Ctrl-C/V-vel átmásolható.

A megoldás kulcsa az OLE (Object Linking and Embedding) technológia, ami bekerült a CMMS-be. A VISION pedig a szokásos módon az OLE objektumon keresztül hivatkozik rá:


```
OC1:OC 123.125,160.75,"CMMS"  
BEGIN  
  Position=      35.25,0.5  
END
```

Ennyire egyszerű a használata, be kell csak írni az OLE szerver nevét: CMMS. Megjegyzem, ugyanezzel a technikával más programok is beágyazhatók a VISION-be (nemrég egy 3D grafikus rajzoló/megjelenítő programot tettünk így a rendszer részévé).

X10 Verzió 1.2

1. Egyedileg skálázható, méretezhető ablakok (1.2)

Az ablakok, képek skálázására eddig csak együtt, az egész alkalmazás részként volt lehetőség, mivel az összes képet ugyanabban a projekt-felbontásban rajzoltuk fel, vagy tettük a befoglaló abkak méretétől függővé – egységesen. Mostantól azonban külön-külön állíthatjuk.

Egy új képtulajdonság teszi lehetővé: *Ablakhoz igazított (FitToWindow)* Egyaránt alkalmazható **operátori képekre** és **ablakokra**! Amennyiben ezt bejelöljük, az adott kép vagy ablak automatikusan változtatja majd a benne látható rajz méretét, ahogy nagyítjuk v. kicsinyítjük.

A képskálázásnak továbbra is két, alapvető módja létezik:

- Rögzített léptékű, szélekhez kötött skálázás
- Arányos skálázás

Megjelenés	
Skála	0,0,PrjWidth/8,PrjHe
Szín	124
Háttérszín	125
Képköltés	SolidFill
Fix skála	False
Ablakhoz igazított	True
Ciklusidő	100
Animációs ciklus	30

Az utóbbi az alap, minden további speciális beállítás nélkül a program arányos skálázást végez. Mégpedig úgy, hogy a változó ablaktartalmat az ablakon belüli objektumok arányos átméretezésével és átpozicionálásával végzi. Ez a szokásos technika.

Kötés	akLeft+akTop+akRight+akBottom
Balra	☒ akLeft
Felülre	☒ akTop
Jobbra	☒ akRight
Lefelé	☒ akBottom

A VISION-ben azonban létezik már egy ideje a projekt felbontásától független képbeállítás, amit a program egyszerűen úgy old meg, hogy a képskálába $PrjWidth / 8$, $PrjHeight / 8$ -at ír. Ezáltal az 1.0 skálaérték mindig 8 pixelnek felel meg – az objektumok pedig nem változtatják meg a méretüket a projekt felbontásának megváltoztatásával.

Ebben az esetben is van azonban értelme a skálázásnak – *szélekhez kötéssel*. Ha pl. az ablak belsejébe egy trendet helyezünk el és azt minden szélhez hozzákötjük (mint fent), akkor az ablak nagyításával a trend szélektől vett távolsága marad változatlan (szemben az arányos skálázással), és ezáltal az ablak belsejében a trend felnagyítódik. Ez a technika általában szebb eredményt ad, mint az arányos skálázás. Cserébe némi munkával jár: szigorúan minden egyes objektumunkat kötni kell valamelyik ablakszélhez.

A rögzített léptékű rajzolásnak ablakok esetén van még egy specialitása. Annak érdekében, hogy az ablakokat is függetlenné tegyük a felbontástól, **meg kell jelölnünk a befoglaló objektumot**, ami az ablak méretét meghatározza. Ez általában egy bár, amibe a további objektumainkat belerajzoljuk. A "megjelölés" úgy történik, hogy a befoglaló objektumot a *Container* csoporthoz rendeljük:

```
BR1:BR WindowWidth-86.875,WindowHeight-56.375
BEGIN
  Position=      25.375,25.375
  Color=         MenuColor
  FrameColor=    MenuColor
  Anchor=        akLeft+akTop+akRight+akBottom
  Group=         "Container"
END
```

Ennek alapján tudja eldönteni a program, hogy mihez kell igazítani a többi objektumot, ill. mi az ablak default mérete. Ez a beállítás csak ablakok esetén kell, és azért szükséges, mert nem mindig engedjük automatikusan állítani a képskálát (*FixScale*), a projekt-felbontás pedig változik.

Még annyit, hogy ablakok esetén már korábban is létezett a skálázhatóság opció (flag: *wScalable*). Ez továbbra is él, az új *FitToWindow* beállítással VAGY kapcsolatban áll. Az ablak-skálázás nem új tulajdonság tehát, csak a képskálázás, ill. az ablakok nagyítása szélekhez kötéssel.

Felhasználása: a *TrendWindow* új változata átméretezhető (ld. a következő pontot).

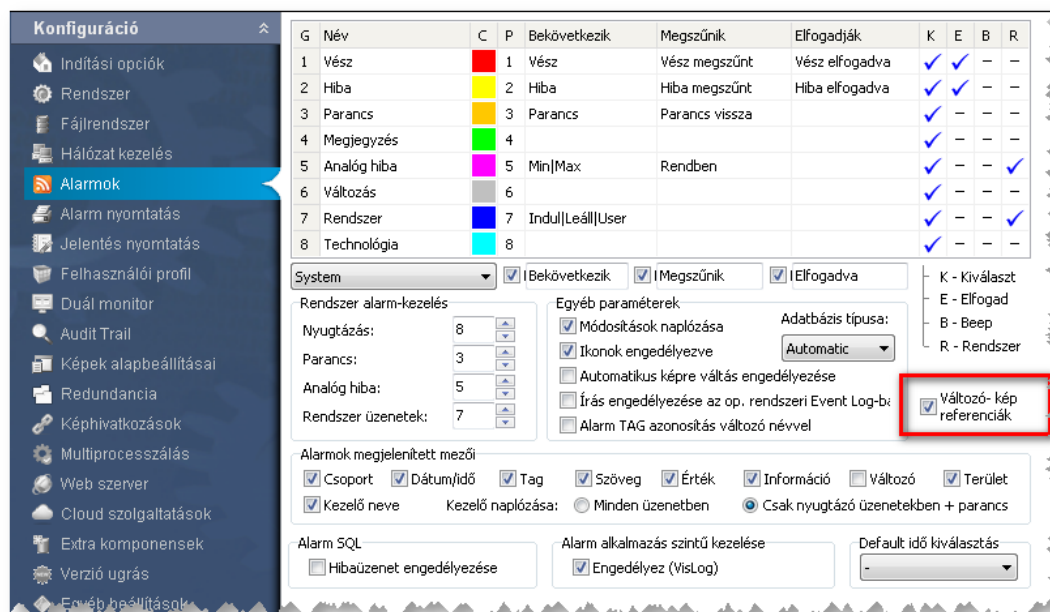
3. Új alarm függvények (1.2)

Két új függvény segítségével lekérdezhetjük, hogy egy képen vagy egy képstruktúrában van-e valamilyenhiba:

```
ImageAlarm::      ushort function (ImageName: shortstring)
ImageStrucAlarm:: ushort function (ID: shortstring; StrucName: shortstring)
```

Az előbbi egy konkrét képre kérdez rá, amit az argumentumban adunk meg, a másik egy képstruktúrára. Ez utóbbi esetben nem a kép nevét használjuk, hanem a kép struktúrabeli azonosítóját. Persze meg kell ilyenkor adni a referencia struktúra nevét is (2. argumentum). A visszaadott érték minden esetben a képen szereplő változók alarm bájtjainak az OR kapcsolata. Az alarm bájt 0-ik bitje az 1-es alarm csoportnak felel meg, az 1-ső bitje a 2-es alarm csoportnak stb.

Ahhoz, hogy a fenti két függvény jól működjön, szükséges még, hogy az alarm konfigurációnál a kép-változó referenciákat engedélyezzük (nagy rendszerben a referenciák előállítása időigényes feladat, ezért nem automatikus, hogy a program effajta nyilvántartást készítsen magáról):



Végül van még egy hasznos függvény, ez azonban nem XSDL-ben lett implementálva, hanem SQL függvényként: *ImageVariables*. Ez a képen látható változókat kérdezi le és helyezi SQL szempontból helyes szintaktikai sorba: (*Variable='VAR1' or Variable='VAR2'...*).

A függvény lényegében kizárólag az alarm napló szűrt tartalmának az előállítására szolgál, hogy a SysLog csak a képhez kapcsolódó hibákat mutassa. A következő példa a tipikus felhasználást szemlélteti:

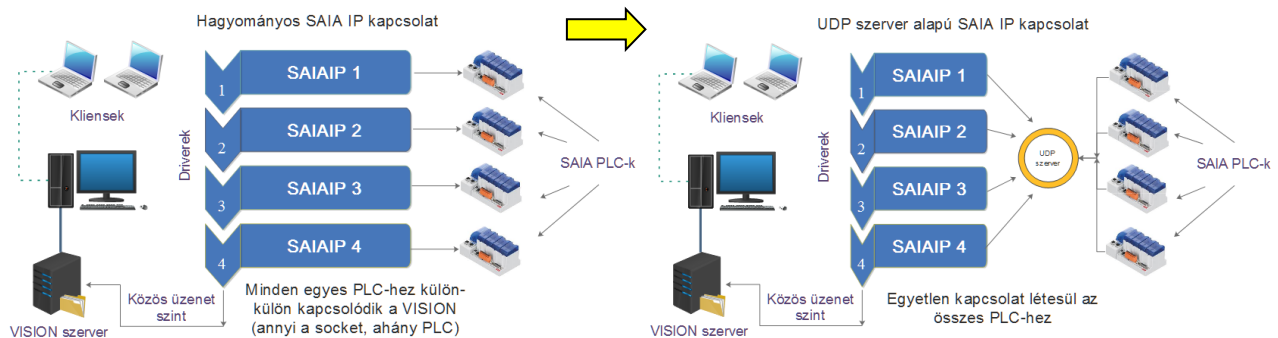
```
sql "VisLog.Log", select, "where ImageVariables",
    "Datum>="+SQLDateTime("VisLog", PackStartOfDay)+" and Datum<"+SQLDateTime("VisLog", CurrentPackTime),
    "order by Datum desc";
```

Mint látható, az *ImageVariables* lényegében egy *where* feltétel függvény. Egyébként a fenti példa az adott nap szűrt eseményeit állítja elő a képre lokalizálva. Érdekes magár az alarm VX objektumot ilyenkor a template-re helyezni, a fenti univerzális SQL parancsot pedig a template *Initialize* ágába.

4. UDP szerver (1.2)

A SAIA és az OMRON kommunikáció egy újabb opcióval bővült: közös UDP szerverrel. Ahehelyett, hogy az egyes driverek külön-külön csatlakoznának a PLC-khez, az új rendszerben ezt a feladatot egy beépített közös UDP szerver végzi el. Ez azt eredményezi, hogy a kapcsolathoz mostantól csupán 1 socket szükséges – akárhány PLC-nk legyen is – a többi már memórián belüli dispatch, szétválogatás.

A változtatást az alábbi két ábra magyarázza:



A változtatás a következő előnyökkel jár:

- Sokkal gyorsabb lett a kommunikáció, amely képes akár 150 üzenet másodpercenkénti adására / vételére
- Nem korlátozott többé a kommunikációs driverek száma, ill. a korlát valahol nagyon magasan jelentkezik, 2 Gbyte körüli össz driver memória igénynél, és a Windows memória kezelés az oka
- Megbízhatóbb lett, mivel csak 1 UDP socketet kell fenntartani és a működését garantálni

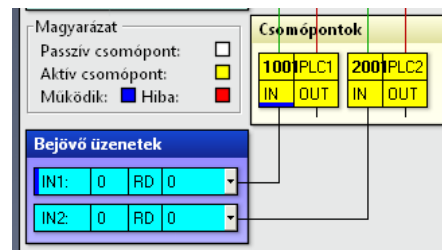
A korábbi változatnál az okozta a gondot, hogy korlátlan számú socket, ami korábban driverenként külön-külön létesült, nem hozható létre egy Windows szerveren belül, másrészt bizonyos szám felett – például 500 PLC esetén – a rendszer már lassulni kezdett. A változtatással mindez megszűnt!

A másik előny, hogy abban az esetben, ha a kapcsolat egyik-másik PLC-vel bizonytalan, az egyáltalán nem befolyásolja a többi működését, mivel a kommunikáció valójában memórián belüli szétmásolásra egyszerűsödött, a közös UDP socket pedig nem törődik azzal, ha nem érkezik válasz valamelyik PLC-ből. Ezt a jelenséget amúgy csak a 64-bites rendszerben tapasztaltuk, a 32-biten nem, de a változtatással ez is megszűnt – a rendszer minden szempontból jobb és gyorsabb lett, 32- és 64-biten egyaránt.

Az UDP szerver alapú kommunikációhoz minden egyes üzenetnek egyedi sorszámot kell adni IP-címenként (csomópontonként) legalább, de a legjobb, ha minden üzenetsorszám egyedi (0..65535).

Az OMRON UDP alapú kommunikációs driverének megjelent egy új változata, OMRONUDPNY néven. Ez a driver arra képes, hogy ugyanazon a csomópont címen egyetlen driver is elég legyen akár tucatnyi PLC-vel kommunikálni abban az esetben is, ha az egyes végpontok IP címben és / vagy port címben térnek csak el egymástól).

A megoldás lényege, hogy a csomópont számhoz 1000-et hozzá kell adni az első PLC-nél, 2000-et a másodiknál stb. (div 1000 lesz a PLC sorszáma).



X10 Verzió 1.1

1. Integrált felhasználó kezelés (1.1)

A módosítás két részből áll: egyrészt áttérhetünk mostantól SQL adatbázisra a felhasználói adatbázis esetén is, másrészt kibővült a felhasználói adatbázis a kapcsolódó termékek felhasználói adataival. Konkrétan a CMMS karbantartás menedzsment és az Energrade energiamenedzsment rendszer lett beépítve a VISION mellé/ alá. A módosításnak egyszerű a magyarázata: egyre több olyan rendszer készül, ahol ennek a három rendszernek – valamilyen kombinációban – együtt kell működnie, és nyilván kézenfekvő, hogy a felhasználók adatait ne kelljen egyik rendszerből a másikba átmásolni, vagy ami még rosszabb, a felhasználót itt is, ott is felvenni. Az új konfiguráló ablak látható alább:

Felhasználó kezelés

Felhasználók: Jogosultságok Szolgáltatások Opciók Hatóság Forrás

Teljes név: * Etelközi Dezső Szervezet: MBI Felhaszn. ID: 2
 Login név: * etelkoz Telefon: +36209087612
 Kulcsszó: * Cím: 1047 Budapest, Batthyány u. 19.
 Kulcsszó ismétlés: Email: provicon@provicon.hu
 Nyelv: Hungarian Megjegyzés:
 Jogosultság: Operator Extra beáll.: ☐ SNMP ☐ Active Directory-ből örökölt

Jelszó politika: ☐ Erős ☐ Jelszó módosítása első belépéskor

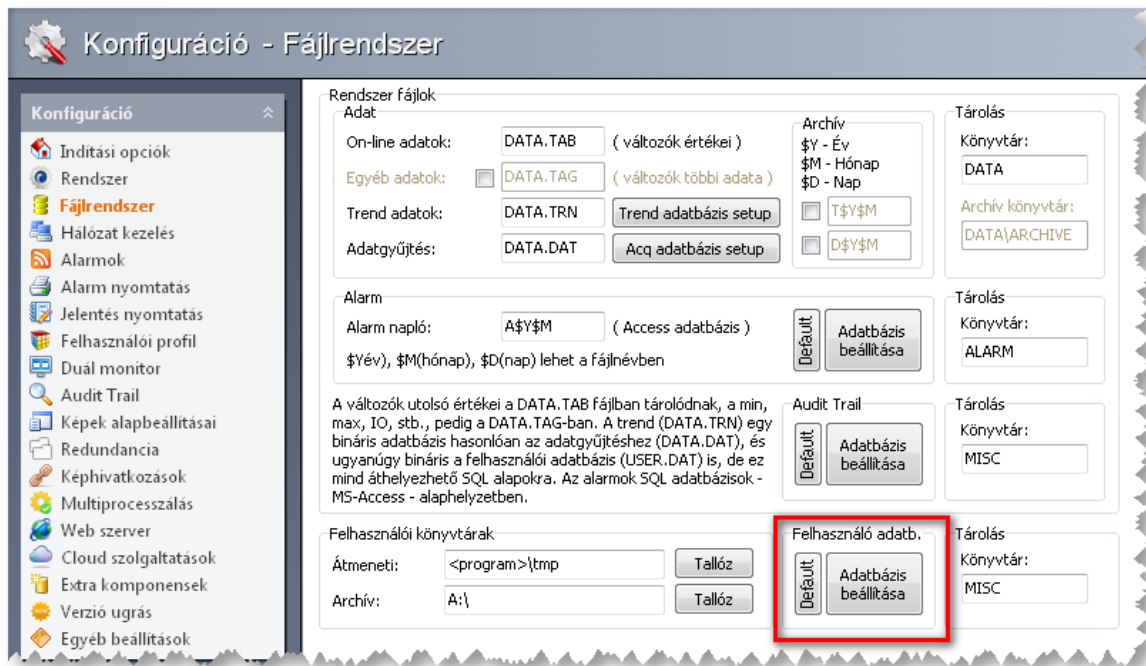
További felhasználói adatlapok:
☒ Részletek ☐ CMMS ☐ EMS ☒ E lapok engedélyezése mindenkinek

Extra beállítások másolása vágólappal:

UID	Teljes user név	Login név	Kulcsszó	Szint	Jogok	Nyelv	Szervezet	Email
1	Adminisztrátor	admin	*****	Administrator	0	Hungarian	MBI	provicon@provicon.hu
2	Etelközi Dezső	etelkoz	*****	Operator	0	Hungarian	MBI	provicon@provicon.hu
3	Lakatos János	lakatos	*****	Operator	0	Hungarian	MBI	provicon@provicon.hu
4	Fejetlen Péter	fejetlen	*****	Tiltott	0	Hungarian	MBI	provicon@provicon.hu
5	Kisfaludi Gábor	kisfalud	*****	Operator	0	Hungarian	MBI	provicon@provicon.hu
6	Domonkos István	domonkos	*****	Operator	0	Hungarian	MBI	provicon@provicon.hu
7	Tőpörtyű Zoltán	topor	*****	Administrator	0	Hungarian	MBI	provicon@provicon.hu
8	Lelkes Csaba	lelkes	*****	Tiltott	0	Hungarian	MBI	provicon@provicon.hu
9	Nagy Menő Benő	nagy	*****	Administrator	0	Hungarian	MBI	provicon@provicon.hu
10	Kerepes Tarcsa	kerepes	*****	Operator	0	Hungarian	MBI	provicon@provicon.hu

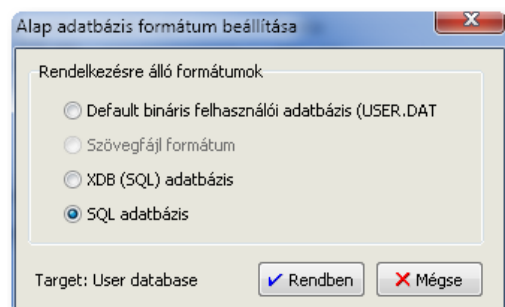
A szerkesztő ablak teljesen eltér a korábbitól és látszólag jóval bonyolultabb annál. De valójában ez szabályozható. Amennyiben a kép közepén található *További felhasználói adatlapok* alatt a *Részleteket* kikapcsoljuk, visszakapjuk a korábbi egyszerű formátumot (azzal az apró különbséggel, hogy most is *head-line* adatbázis szerkesztő felületet kezelünk). Ugyanitt található meg a *CMMS* és az *EMS* (energiamenedzsment) adatainak a bekapcsolási lehetősége. Tehát az említett plusz adatok csak akkor válnak elérhetővé és szerkeszthetővé, ha arról külön rendelkezünk. Ráadásul felhasználónként akár eltérő módon: lehetnek olyan felhasználóink, akik csak a VISION-höz szükséges alapadatokat láthatják és szerkesztik, mások a CMMS-t is, megint mások az EMS-sel együtt akár mindhármat.

Ami pedig az SQL-re való áttérést illeti, az alarmoknál és az Audit Trail-nél megszokott eljárást követjük. A konfigurációs ablak kibővült egy új menüponttal:



Erre kattintva a jobb oldalt látható, már a korábbi adatbázis beállításokból is ismert SQL választó dialógus jön elő, ahol választhatunk MS-SQL (Oracle stb.) és XDB alapú SQL adatbázis kezelés között, amit a következő pontban ismertetünk.

Az áttérés során a program megkérdezi, hogy szeretnénk-e átkonvertálni a korábbi adatokat az új adatbázis alá, aminek a tábláit persze létre is hozza, de ezt bármikor később is megtehetjük, amikor adatbázist váltunk, sőt visszatérünk a *Default bináris felhasználói adatbázisra* (USER.DAT). De természetesen a többlet adataink ilyenkor elvesznek. További különbség, hogy SQL adatbázis alatt mások a felhasználónévre, jelszóra stb. vonatkozó korlátok. Például a maximális karakterszám a jelszóra 32 (korábban 11 volt).

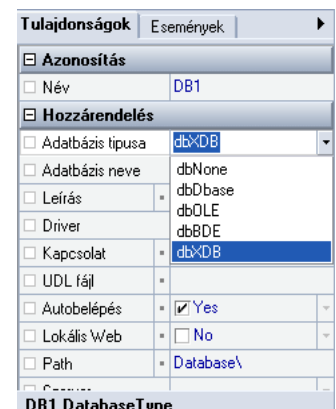


2. Beépített SQL adatbázis kezelő (1.1)

A VISION.64 15-ös változata egy régen várt szolgáltatással bővült: **saját SQL adatbázis kezeléssel**. Nem feltétlenül kell többé MS-SQL-el, Access-szel vagy mySQL-el bajlódni, a VISION saját SQL kezelője is ellátja ugyanazokat a funkciókat, ráadásul a felsoroltaknál **sokkal gyorsabban!** Az időközben elavult DBASE mellett tehát megjelent egy további natív SQL motor a VISION-ben, és ennek megfelelően bővült a választható adatbázis típusok listája. Ez látszik jobb oldalt. A korábbi *dbVision* bejegyzés helyére *dbDbase* került (mivel valójában az volt), aminek ugyanaz a kódja, mint a *dbVision*-é volt, így a rendszer megőrizte kompatibilitását a DBASE-t használó rendszerek felé (*dbVision* = *dbDbase*).

Új azonban a **dbXDB** adatbázis típus, ami a nevéhez híven az XDB kiterjesztésű, már korábban is elterjedten használt szövegfájlokra épül. Az XDB alapú adatbázis kezelés elve a következő:

Az adatbázis egy-egy konkrét könyvtárt azonosít, a táblák az ott található XDB fájlok. Az adatbázis könyvtár elérési útját a *Path* paraméter határozza meg a képen látható módon (*Database*). Lehet



relatív vagy abszolút, a VISION telepítési könyvtára alatti vagy azon kívül mutató – ahogy tetszik. Az adatbázis nevét választhatjuk azonosra ezzel a könyvtár névvel. Minden egyes XDB fájl, ami ebben a könyvtárban van, mint tábla automatikusan része az XDB adatbázisnak.

Minden más – mezők típusai, ábrázolás módja, változó hozzárendelés, cloud, hálózati átvitel, SQL függvények stb. – megegyezik az SQL kezelésnél megismert megoldásokkal. Létrehozhatunk, törölhetünk táblákat, benne mezőket, rendezhetjük az eredményhalmazokat bármely mező szerint növekvő v. csökkenő sorrendben, tetszőlegesen bonyolult *where* feltétel részt használhatunk és változó mezőnév sorrendet. Ahogy megszoktuk pl. MS-SQL alatt. Sőt, az átállást megkönnyítendő az új XDB adatbázis kezelő ismeri az összes MS-SQL, Access és MySQL függvényt is! Ha tehát valaki be is égette a kódba mondjuk az MS-Access-t, XDB változattal is működni fog a programja anélkül, hogy azon változtatni kellene. Meglévő SQL adatbázisok az

fh "DATA.TABLE",ConvertXDB,"Database\Table.xdb";

paranccsal kovertálhatók át XDB táblába. Maga az XDB fájl egy átnevezett kiterjesztésű CSV fájl azzal az egyetlen apró különbséggel, hogy az első sor tartalmazza a mezőneveket, a mezőnevek mellett szögletes zárójelben pedig opcionálisan a mező típusát és méretét. Elválasztó karakterként a pontos vessző mellett inkább az oszlop | karaktert javasoljuk, hogy a szöveg belsejében található ; karakterek ne okozzanak gondot. De használható a pontos vessző is, sőt megadhatunk saját elválasztó karaktert.

Az XDB alapú adatbázis kezelés azért nagyon gyors, mert szövegfájlokra alapul. A védettsége nyilván nem éri el egy MS-SQL vagy Oracle alapú rendszerét, cserébe viszont nagyon könnyű kezelni, mozgatni, archiválni a fájlokat, vagy visszamásolni egy mentett változatból, és kézzel vagy Excel-el is módosítható. Az alkalmazás saját fájljain túl alarm kezelésre és az Audit adatbázisra javasoljuk a használatát.

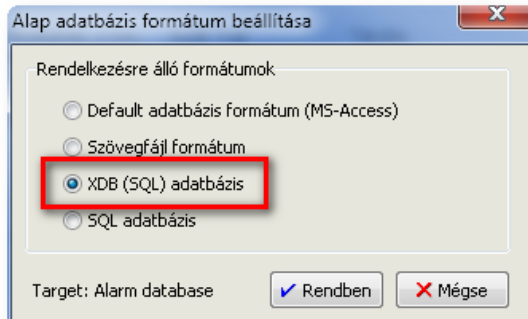
Miért volt szükség saját SQL motorra? Elsősorban a 64-biten nem igazán használható MS-Access miatt. Speciális adatbázis motor, pl. MS-SQL telepítése nélkül is megoldható mostantól az alarm kezelés, sőt a korábbi naplókat is áttehetjük ebbe a formátumba – mégéri. Mivel sokkal gyorsabb, egyszerűbb és az Access-nél legalább is biztosan megbízhatóbb!

Az XDB adatbázisnak nincsenek méret korlátai, és kisebb adatbázisok esetén akár 20-szor(!) gyorsabb lehet a Microsoftnál. Nagyobbaknál (több százezer, millió rekord), lényegében az MS-SQL teljesítményét reprodukálja.

Előnyei miatt betettük e szolgáltatást a cloud szervízekbe, így a PVCS szervízbe is.

2.1. Áttérés XDB alapú rendszer adatbázis kezelésre

Az előzőekben vázolt megoldás a rendszer adatbázisok szintjén is elérhető. Mostantól, amikor a *Konfig* ⇒ *Fájlrendszer* alatt az adatbázis kapcsolat menüpontok bármelyikét kiválasztjuk, az alábbi, módosult dialóg ablak jelenik meg:



A mutatott *XDB (SQL) adatbázis* sor az új. Ezt kiválasztva térhetünk át XDB adatbázis kezelésre a hagyományos SQL és a szövegfájl formátum mellett.

Itt jegyzem meg, hogy szövegfájl formátum korábban is létezett pl. az alarmok kezelésére, de az messze nem ugyanaz, mint az XDB, lévén ez utóbbi valódi SQL adatbázis, amire ugyanolyan Query-k értelmezettek, mint pl. az MS-SQL-re. Létre lehet hozni ilyen táblákat pl. az sql Create paranccsal, míg szövegfájlt nem.

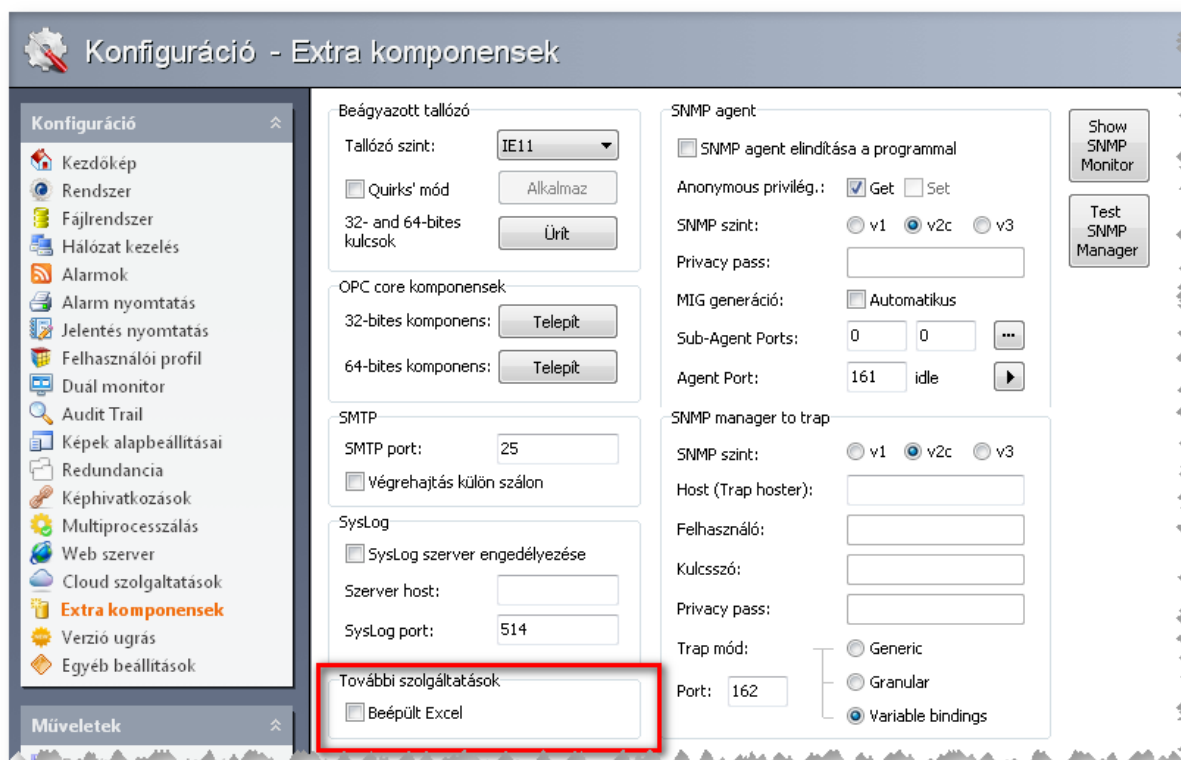
3. Extra szolgáltatások

Tekintettel a szolgáltatások nagy számára és arra a körülményre, hogy mindenkinek biztosan nincs szüksége minden szolgáltatásra, az extra szolgáltatásokat külön licence-hez kötöttük, az árakat ezáltal jobban széthúzhattuk. A megoldás alapja egy szolgáltatás leíró fájl, amit a program szolgáltatójától lehet megrendelni az árlista alapján és a tőle megkapott, adott VISION licencehez rendelt fájlt a VISION-ben regisztrálni. Ehhez a művelethez nem szükséges semmilyen speciális program, nincs hozzá menüpont a VISION-ben; akkor aktiválódik, amikor a kívánt szolgáltatást a konfigurációs lapok valamelyikén be akarjuk kapcsolni. Ilyenkor, ha az adott szolgáltatás még nincs regisztrálva, a VISION automatikusan bekéri a szükséges szolgáltatás leíró fájlt és a Bin alatt található Services.xml fájlba beírja. Felhívjuk a figyelmet, hogy ez a fájl nem szerkeszthető kézzel, mivel önmagában ellenőrző összeget tartalmaz a fájl tartalmára és a VISION csak akkor fogadja el, ha ez stimmel!

4. Beépült Excel (1.1)

A programhoz a továbbiakban nem szükséges többé az Office csomag, ha csak Excel fájlokat szeretnénk kezelni, abba adatokat exportálni, vagy xls fájlokból adatokat kiolvasni, mivel egy beépült, teljes értékű Excel komponens ezt Office nélkül is megoldja. Ez ugyanakkor licence-el szolgáltatás (ld.1. pont).

Engedélyezése az Extra komponensek lapon lehetséges:

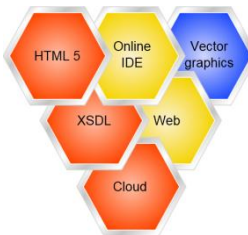


Amint engedélyezzük, az összes Excel műveletet átveszi a beépült komponens. Az OLE funkciót is! A beépülő komponens egy nagyságrenddel gyorsabb az Excel-nél és többlettudással bír a cellák színezését, formattálását illetően is. Mindenképp javasolt a használata, különösen nagyszámú sort tartalmazó Excel export esetén.

5. SVG (Scalable Vector Graphics) (1.1)

Egy különleges szolgáltatással bővült a VISION: SVG fájl megjelenítéssel. Ez a fájl típus ugyanúgy LD paranccsal rajzolható ki a képre, mint egy BMP, PNG vagy JPG fájl, azonban nagy különbség, hogy **vektorgrafikusan!**

Az SVG de facto szabvány a vektorgrafikus ábrázolásban. Valójában egy XML fájl, ami szövegesen tartalmazza a grafika felrajzolásához szükséges metadátákat. Az SVG-t eredetileg Web-es tartalmak vektoros előállítására fejlesztették ki, a szabvány a W3C által lett definiálva és publikálva. Ma minden modern tallózó támogatja közvetlen rajzelemként és HTML tag-ként egyaránt, vagyis közvetlenül az oldalba ágyazva. A Web-en tehát egyértelműen ez az irány a szabvány 2003-as megjelenése óta. A HTML 5 tudással rendelkező böngészők mind ismerik – inkább kisebb, mint nagyobb hibákkal.



Nem teljesen triviális azonban ennek a fájl típusnak a használata desktop környezetben – kapásból az Office sem ismeri –, mivel kevés komponens képes rá, hogy az oldallal együtt (tehát ne egy önálló ablakban) renderelje. Másrészt egyáltalán nem egyszerű az implementálása sem, mivel az SVG bonyolult mátrix műveleteket, szimbólumokat, sőt CSS3 utasításokat is tartalmazhat, valamint layer megadási lehetőséget, azonosítókat, szimbólumokat stb. **Valójában egy mini böngésző implementálás**

szükséges hozzá. A VISION fejlesztése azonban nem a meghátrálásról híres, ezért az SVG szabványt forrás szinten beépítettük a rendszerbe, sőt az SVG tartalmat a program XSDL grafikus objektumokra fordítja le, s így az X10-es rendszerben az SVG a saját grafikus editorával szerkeszthető. Azonban már a VISION X9.64-es rendszerben is élvezhetjük a szolgáltatásait önálló grafikus rajzelemként.

Az SVG számtalan előnyéből a legfontosabb, hogy szinte minden vektorgrafikus rendszerből exportálhatók grafikák SVG-be (AutoCAD, CorelDraw, Edraw, Microsoft Visio stb.), vagyis ezekben az alkalmazásokban készült képek minőségromlás nélkül tehetők be a VISION-be. Az Office és általában a Microsoft vonalon pedig a saját vektorgrafikus WMF és EMF formátumából konvertálhatunk SVG-be képeket – például az **emf2vec** programmal, vagy az elterjedten használt **inkscape**-pel. Így bár az Office közvetlenül nem támogatja, mégsem probléma SVG-be konvertálni a képeit.

Érdekes itt megjegyezni, hogy az ablak context management szintjén a VISION már a **HTML 5 canvas**-t használja, pontosan ugyanazokat az utasításokat, amit a Web oldalak is a dinamikus vagy csak simán vektoros tartalmak előállítására. Olyannyira, hogy magát a HTML 5 scrip-et is röptében legyártja! Hogy kapcsolódik mindez a témához? Konkrétan ez történik az SVG kirajzolása közben is. A VISION pont úgy jár el, mint bármelyik böngésző. Ennek a megoldásnak következménye, hogy akár a script is közvetlenül Web oldalba tölthető, ami ezáltal ugyanazt a dinamikus tartalmat mutatja, mint a VISION. A példaként megmutatott SVG VISION embléma a jobb oldalt látható scriptet generálja (részletét láthatjuk itt).

```
ctx.save();
ctx.strokeStyle="rgba(0,0,0,0)";
ctx.miterLimit=4;
ctx.font="normal normal normal normal 15px / 21.4286px """;
ctx.font=" 15px ";
ctx.scale(0.3404255319148936,0.3404255319148936);
ctx.save();
ctx.font=" 15px ";
ctx.save();
var g = ctx.createLinearGradient(3.5,-3.5,138.5,131.5);
g.addColorStop(0,"rgb(49.803922%,49.803922%,49.803922%)");
g.addColorStop(0.18,"rgb(100%,100%,100%)");
g.addColorStop(0.32,"rgb(76.862745%,76.862745%,76.862745%)");
g.addColorStop(0.52,"rgb(100%,100%,100%)");
g.addColorStop(0.68,"rgb(76.862745%,76.862745%,76.862745%)");
g.addColorStop(0.85,"rgb(100%,100%,100%)");
g.addColorStop(1,"rgb(76.862745%,76.862745%,76.862745%)");
ctx.fillStyle=g;
```

Felhívjuk a figyelmet, hogy az SVG-nek sok változata létezik és egyetlen implementációja sem tökéletes, azaz nem minden SVG fájl jelenik meg mindenhol. Pl. az IE hibásan kezeli a szövegeket, számos implementációból hiányzik a clipping, a szövegdekoráció és sorolhatnánk. A VISION sem tökéletes. Az eddig általunk használt SVG fájloknak kb. a 95%-a volt hibátlan, de volt olyan, ami nem volt az. Ezeket SVG11 opcióval lehet megjeleníteni, és sajnos emiatt kb. 20-szor lassabban. De működnek azok is.

Az SVG a HTML 5 alapú X10 fejlesztés fontos állomása, egyben ki is jelöli a stratégiai irányt, ami jelen állás szerint egybeesik a grafikus rendszerek fejlesztési irányával a világban. Különösen azt hozzátéve, hogy időközben megszületett a szabvány 3D változata is.

6. Update management (1.1)

A program kibővült egy alkalmazás szintű update management szolgáltatással. A VISION program egy ideje tartalmaz már automatikus update szolgáltatást, ez azonban magára a VISION rendszerre korlátozódott, azaz nem tartalmazta a felhasználó saját alkalmazását. Ami nyilvánvaló, hiszen az update-et a VISION program kibocsátója, többnyire a Provicon Kft. állította elő, így az nem tartalmazhatott felhasználó által fejlesztett elemeket.

A most készült bővítmény ez utóbbit teszi lehetővé az alkalmazás fejlesztők számára. Egyszerűen egy új paranccsal maga az alkalmazás készíthet magáról VCB kiterjesztésű csomagokat és ez opcionálisan tartalmazhatja a rendszerprogramokat, valamint az alkalmazást is. Ezt a fájlrendszert a kliensek lefrissíthetik a lokalizált, saját állományukba. A megoldás alapja a következő, új VISION utasítás:

```
CreateCabinetFiles:: procedure (CabinetName: optional namestring;  
                                Parts: optional ushort;  
                                PackageName: optional shortstring;  
                                PackageVersion: optional shortstring;  
                                Publisher: optional namestring;  
                                files: optional list of string)
```

Az első opcionális paraméter határozza meg a cabinet file nevét, default értéke *ProductSheet*. Ez az elnevezés egyébként a VISION update csomagok neve, azt nyugodtan felülírhatjuk. De megadhatunk itt saját nevet is. A *Parts* paraméter szolgál annak meghatározására, hogy a rendszert és az alkalmazást, ill. az alkalmazásokat is bevonjuk-e a csomagba.

Parts: 0.bit = rendszer;
 1.bit = aktuális alkalmazás;
 2.bit = összes alkalmazás a szuperprojektben (beleértve az aktuálisat)
 Default értéke = 5

Az utasítás többi része – *PackageName*, *PackageVersion* és *Publisher* önmagáért beszél, bármilyen szövegeket megadhatunk itt. A *PackageVersion*-nek, azaz a verziószámnak azonban lehet még máshol is szerepe (felhasználó tájékoztatása a verziószámról).

A parancs utolsó opcionális paramétere lehetővé teszi további fájlok vagy könyvtárak becsomagolását a cabinet-be. Ez olyan speciális bitmap-ek esetén szükséges pl., amit a program nem tud magától kitalálni. Hogy jobban érthető legyen, képzeljük el, hogy alkalmazásunk változóban megadott bitmap-re hivatkozik a képen. Vagyis futási időben dől el, hogy mit akarunk kirajzolni. Ilyen esetben a program a scriptet olvasva nem tudja kitalálni, hogy mit akarunk, ezért azt a könyvtárat, ahol a lehetséges képeket őrizzük, ilyenkor külön meg kell adni. Egyébként a program ugyanazt az eljárást használja a fájlrendszer feltérképezésére, mint a lokalizált kliens, az pedig a fenti speciális eseteket leszámítva valóban tökéletes eredményt ad, miközben csak a szükséges fájlokat viszi át a csomagba.

A művelet eredménye egy *<CabinetName>.vcb* file lesz, amit a program a telepítési *Bin* könyvtárba ment. (Tehetjük máshová is, ha megadjuk a teljes elérési utat, de nem célszerű, mivel a rendszerfájlokat is érinti a művelet.) Eredetileg ott egy *ProductSheet.vcb* fájl található, ami a telepített VISION rendszer saját cabinet fájlja, benne a demóval, de azt simán felülírhatjuk, hisz csak a lokális update szolgáltatás használja.

A parancs pár másodperc alatt lefut, tehát akár ciklikusan is kiadható naponta mondjuk egyszer, vagy kiteve egy gombra. De nyilván módosítás és ellenőrzés után érdemes csak új csomagot készíteni.

Maga az update művelet a lokalizált üzemmódú kliens program újraindításakor történik meg és a kis betöltő program végzi a következőképpen: amikor újraindítjuk a VISION-t, a program bekéri a szerverből a megadott cabinet fájlt, majd megnézi, hogy az ottani fájllistában található-e újabb a lokális fájlrendszerben lévő változatoknál, majd az új fájlokat letölti a szerverről. A szerveren cabinet-be csomagolt és a lokális fájlrendszer így végül azonos lesz.

Mivel update csak újraindítással lehetséges, felmerül a kérdés, honnan tudhatja a kliens operátora, hogy neki újra kell indítania a programot, azaz hogy update csomag keletkezett? Erre szolgál két további utasítás:

```
GetCurrentVersion:: shortstring function (CabinetName: optional namestring)
GetUpdateVersion:: shortstring function (CabinetName: optional namestring)
```

Az előbbivel az aktuális verziószámot, az utóbbival pedig a szerveren elérhető cabinet verziószámát kérdezhetjük le. Ha e kettő nem azonos, elérhető egy update csomag és akár feltételesen kiírhatjuk mindkét verziószámot a képernyőre. Ezután a felhasználót már csak arra kell megtanítani, hogy ha új verziót lát és a helyzet is lehetővé teszi, indítsa újra a kliens programot. Magát a verziószámot a csomagkészítés során adjuk meg a *PackageVersion* paraméterben. Saját verziókövetést végezhetünk.

Arra vigyázzunk, hogy a *GetUpdateVersion* parancsot ne adjuk ki túl gyakran és feleslegesen, mivel ez ténylegesen lekéri a szerverből az egész csomagot, hogy kiolvassa belőle a verziószámot. Tegyük inkább a ciklikus programba és mondjuk óránként indítsuk el, az eredményt pedig írjuk egy változóba.

6.1. Update konfiguráció

Az előzőekben ismertetett programutasításokon túl az update szolgáltatást konfigurálnunk is kell:

The screenshot shows the 'Konfiguráció - Egyéb beállítások' (Configuration - Other settings) window. The 'Update' section is highlighted with a red box. It contains the following settings:

- Automatikus adat export:**
 - ☐ Adat export engedélyezése
 - Mód: ☒ ciklikus ☐ változásra
 - Ciklusidő: 500 ms
 - Léptetés: 3 változó
 - Tábla: T_Vision
 - Adatbázis: Adatbázis beállítása
- Update:**
 - ☒ Nincs automatikus update, ellenőrzés és letöltés manuálisan
 - ☐ Automatikus update (ellenőrzés és letöltés programindításkor)
 - Update szerver: http://www.provicon.net:9000/updvcb
 - Buttons: Update host keresés, Alkalmaz, Saját update beállítása, Csomagkészítés
- Alkalmazás védelem:**
 - ☐ Engedélyezve
 - Az alkalmazásokat megvédhetjük az illegális használattól kulcsszóval és aktiválással

Other visible settings include:

- Csatolt projekt:** Projekt könyvtár, Csatló projekt engedélyezése, PLC rendszer (Ismeretlen), Változó fájl (xvar), Kommunikációs fájl (xcom).
- Video:** Jpeg dekódér (GT-IJL), Delta frame kicsomagoló (GT-IJL), Dekódolás külön szálon, Új stream, ha átméretez, Videolassítás egyszerűsített módban.
- Fejlesztő (IDE) <-> Futató (Run-Time):** Új kulcsszó, Kulcsszó ismétlés.
- Alkalmazás titkosítás:** Titkosító kulcs.

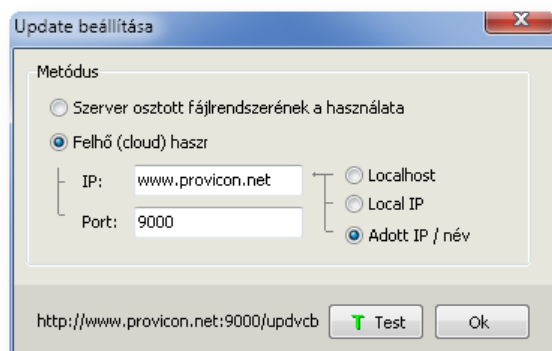
A fent látható módon az update server IP címét és portcímét kell megadnunk, valamint a csomag nevét (*mypack*), ha az nem *ProductSheet* és cloud szolgáltatást használva akarjuk elérni az update-et, vagy magát a fájlt, ha osztott fájlrendszeren keresztül. Cloud szolgáltatás használatához a VISION szerverben a Web szervert is engedélyezni kell, de **használhatunk külső Cloud szervert is, pl. a PVCS programot, hogy ezzel ne terheljük a VISION szervert!**

Amennyiben csak a fájl nevét adtuk meg, a program az osztott fájlrendszer alapú kliens lokalizálást finomítja tovább: ahelyett, hogy folyton ellenőrizné a szerver és a kliens fájlrendszerét induláskor, csak a cabinet-et nézi meg, ami ezért sokkal gyorsabb. Tehát még hagyományos kliens lokalizálás esetén is jobb bekapcsolni az update management szolgáltatást. Cloud esetén pedig az osztott fájlrendszer küszöbölhető ki általa. Az update server megadása mindkét esetben tartalmazza az *updvcb* path-t, ha nem adtuk meg, a program fogja beírni helyettünk. Abban az esetben azonban, ha a csomag nevét is megváltoztattuk az alapértelmezett *ProductSheet*-hez képest (mint fent: *mypack*), akkor szerepeltetni kell.

Néhány lehetséges megadás:

Alapeset	Update szerver	Csomag neve
Cloud	http://192.168.1.1:9000	ProductSheet.vcb
	http://192.168.1.1:9000/updvcb	ProductSheet.vcb
	http://192.168.1.1:9000/updvcb/mypack	mypack.vcb
File share	updvcb	ProductSheet.vcb
	mypack	mypack.vcb
	updvcb\mypack.vcb	mypack.vcb
	updvcb\mypack.myext	mypack.myext

A **kékkel** színezett sorok esetén a program automatikusan szűrja be utólag az *updvcb* specifikációt. (Ami azért szükséges, hogy a cloud szerver el tudja dönteni, mit akar a kliens – ne felejtjük el, van még vagy száz további cloud parancs. A fájlt a telepítési *Bin* könyvtár alatt keresi a program.



A beállítást segíti a fenti konfigurációs oldalon található *saját update beállítás* gomb és a hatására megjelenő dialógus (balra).

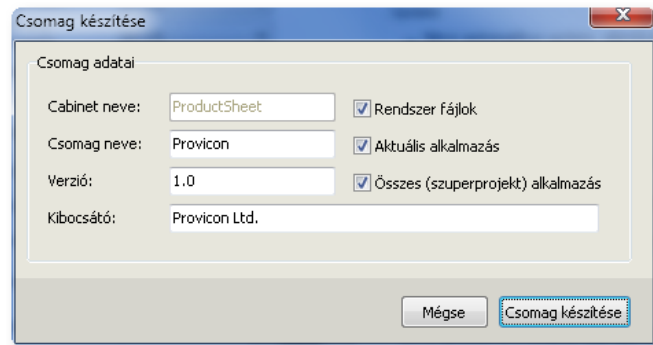
A programocska az URL előállítását segíti; képes lekérdezni a szerver saját IP címét és eldönthetjük, hogy osztott fájlrendszert használjunk-e vagy cloud-ot. A program pedig a fenti specifikáció szerint jár el.

Végül lehetőség van arra is, hogy a saját csomagunkat ne alkalmazói programból, hanem

mindjárt itt, a konfigurációs képen készítsük el.

Erre szolgál a *csomagkészítés* gomb és a megnyomásakor megjelenő következő konfigurációs ablak:

Itt megadható minden információ, amit a programból kiadható utasításnál is láthattunk, az egyedi fájlok és könyvtárak kivételével.



Csomag készítése

Csomag adatai

Cabinet neve: ProductSheet ☒ Rendszer fájlok

Csomag neve: Provicon ☒ Aktuális alkalmazás

Verzió: 1.0 ☒ Összes (szuperprojekt) alkalmazás

Kibocsátó: Provicon Ltd.

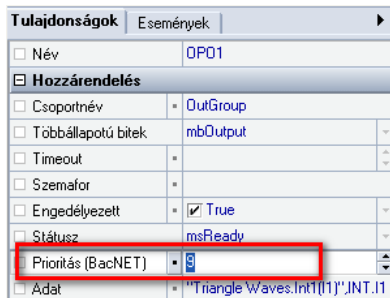
Mégse Csomag készítése

A megoldás legfontosabb előnyei a hagyományos kliens lokalizáláshoz képest:

- Gyorsabb, mivel a szerveren lévő fájlokkal a programnak nem kell összehasonlítania a saját fájlrendszerét (hisz a letöltött csomag pont ezt tartalmazza)
- Jobban kontrollált, hiszen a cabinet készítésével döntjük el, hogy új változat készüljön és nem azzal, hogy új fájlokat másolunk, vagy meglévőeket módosítunk
- Verziókövetést valósíthatunk meg általa
- Cloud alkalmazásával a kliens lokalizálás osztott fájlrendszer nélkül valósul meg
- Cloud esetén a VISION szerver és az update szerver lehet két különböző gép

7. BacNET támogatás (1.1)

A BacNET OPC szinten támogatott a VISION programban. Az OPC régi komponense a VISION-nek, felvetődik tehát a kérdés, mi ebben az új? Nos, a BacNET parancsoknak van egy specialitása: a prioritás kezelés. Amikor egy BacNET OPC node-ra írunk, azt is meg kell adnunk az esetek többségében, hogy milyen prioritási szinten, mivel ezeket a BacNET PLC teljesen eltérően kezeli. Például a manuális output prioritási csoportja a 9-es. Ezt tehát az OPC kimeneti üzenetben meg is kell adnunk:



A valóságban a prioritási csoport az OPC item szöveges értékéhez adódik hozzá, belül:

`"BacNET_PLC_WriteableItem:9"`

A konfigurációs során csak a kétőspont előtti részt vesszük fel, maga az OPC browser csak ezt kínálja fel(!), a szükséges kiegészítést (piros) a VISION teszi hozzá (a beállított és akár változóhoz is hozzá-rendelhető tulajdonság alapján) menet közben.

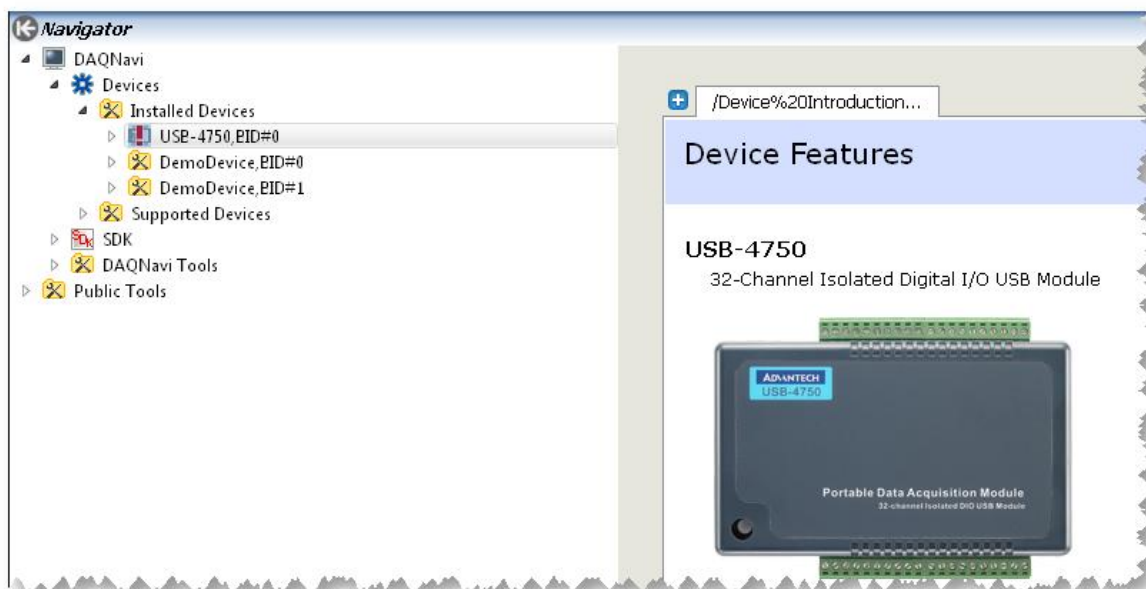
Bővebb információ itt:

https://scadahacker.com/library/Documents/ICS_Protocols/Phoenix%20Contact%20-%20BACnet%20Primer.pdf

7. Advantech DAQ (1.1)

Készült egy driver az Advantech DAQ (Data Acquisition) család kiszolgálására. A DAQ sorozat tartalmazza az összes digitális, analóg és egyéb (pl. WebAccess) eszközt az Advantech kínálatából. Ez a kommunikációnak az általuk preferált, új módja.

Az eszközöket a DAQNav, készülék navigátorral kell felsimertetni és felvenni, és a VISION ezt a konfigurációt veszi át automatikusan. Például:



Az Advantech ezzel igazán korszerű megoldást alkotott, mi pedig örömmel alkalmazzuk és ajánljuk.

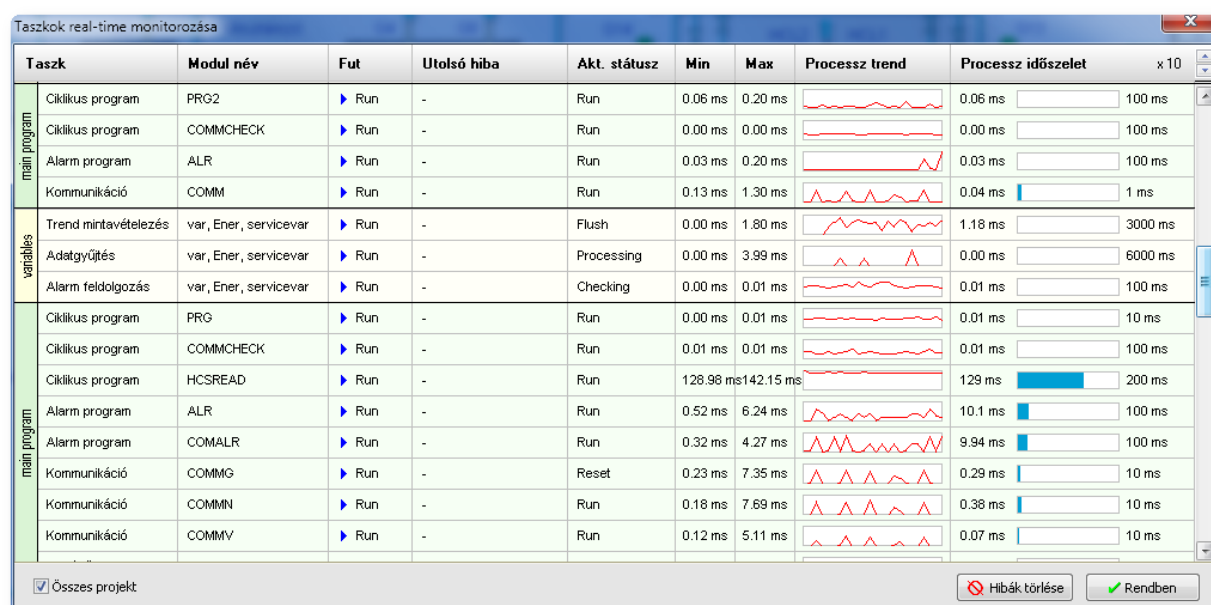
Változtatások az VISION X10-ben, verzió 1.0

A programban rengeteg apró, ám annál fontosabb változtatás történt. A legfontosabbak a következők:

- Taszkok monitorozása
- Trendekkel és adatgyűjtéssel kapcsolatos változtatások
- Változók szűrése, listázása, tömbváltozók megjelenítése
- Szövegváltozók küldése és vétele byte-blokkban
- Verzió lekérdezése
- A hálózati átvitel mindig tartalmazza a tömböket (tömbinterface=true)

1. Taszkok monitorozása

A program a háttérben futó taszkok ellenőrzésére és a futási állapot megjelenítésére egy sokkal kifinomultabb és jóval több információt nyújtó real-time monitorral bővült. Ez látható alább:



A folyamat időszület most is a taszk relatív időfelhasználását mutatja a saját ciklusidejében, azonban sokkal pontosabban, tíz mikroszekundumos felbontásban. Tekintve, hogy a VISION processzek általában rendkívül gyorsak, lehetőség van a tartomány felnagyítására a kép jobb felső sarkában látható átskálázó vezérlővel. Például x10-re állítva azt érjük el, hogy a bárgráf 10-ed akkor tartományt mutat, ezáltal felskálázza 10-szeresére a mutatott értékeket. Ezáltal az alsó tartományban mozgó taszkok időigénye is jól mérhetővé és összehasonlíthatóvá válik, amikor a legnagyobb terhelésű taszkat keressük.

További különbség, hogy nem százalékosan, hanem miliszekundumban mutatjuk az értéket – így a szám is könnyebben értelmezhető. A fenti ábrán az látszik, hogy a demo taszjai tipikusan 10-50 mikroszekundumot igényelnek!

Az időfelhasználás két perces trendje a ciklusidő változását szemlélteti, előtte a *Min* és *Max* értékek a két perces mintasor legkisebb és legnagyobb értékei ismét miliszekundumban.

A futási állapot megjelenítése, a modulnevek és a taszkok leírása megegyezik a korábbival, csak itt tagoltabban láthatjuk őket, külön csoportosítva a változó táblákhoz tartozó taszkokat (világos sárga), a főprogram taszkjait (világos zöld) és a unitok taszkjait (világos kék).

A monitor 100 miliszekundumonként, real-time frissül.

2. Trendekkel és adatgyűjtéssel kapcsolatos változások

A legfontosabb ezzel kapcsolatos tudnivaló, hogy a változók táblánkénti kezelése helyébe közös trend- és adatgyűjtés processz lép. Ebből következően **a trend alapciklusideje az összes tábla legkisebb ciklusideje lesz és az adatgyűjtés is a leggyorsabb ciklushoz igazodik** az összes változóra nézve. Ha pl. valakinek volt egy változó táblája, amiben 1 perc volt az alapciklus, és egy másik, amiben 10 másodperc, akkor az X10 a gyorsabb 10 másodperces alapciklussal fogja mindkettőt kiszolgálni. Természetesen ez nem érinti a mintavételezési időt, csak arról van szó, hogy maga a processz 10 másodpercenként vagy 1 percenként vizsgálja meg, hogy egy változó értékéből mintát kell-e venni.

A változtatás fő oka az optimalizálás és a hozzáigazítás a jelen tendenciáihoz. Manapság gyakran készülnek több tízezer változót tartalmazó rendszerek, amelyeket akár 30 táblára is szétbonthatunk. Az X9-es változatban ez 90 programszálát eredményezett (3 szál – trend, adatgyűjtés, alarm – változó táblánként). De ezt a Windows nem tudja olyan hatékonyan kezelni, mintha kevesebb, nagyobb terhelésű szál lenne. A helyzet pedig eskalálódik, amikor szuperprojekteket hozunk létre – nem ritka, hogy akár 20 projektet is összevonunk. Azt is érdemes tudni, hogy a szálakat a trend- és az adatgyűjtés adatbázisába való íráskor szinkronizálni kellett, mivel sem az SQL provider, sem a közvetlen fájllelés nem teszi azt lehetővé, hogy egyszerre párhuzamosan több szálból hívjuk meg és írjuk. (Emlékeztetném az olvasót a 'Trend save error' típusú hibákra, ami pont ebből a problémából fakadt.)

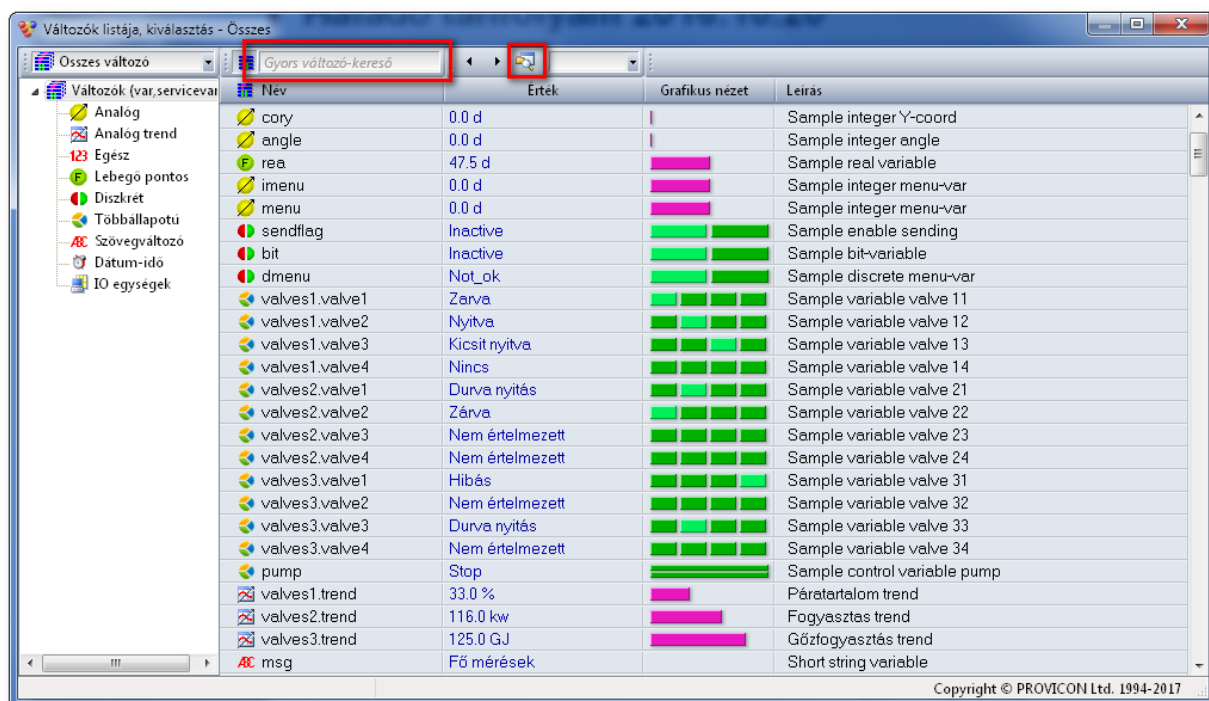
Ezért **a megoldást függetlenítettük az alkalmazás méretétől** projektenként fixen háromra. Szinkronizálni a táblák között mostantól nem kell, hisz egy projekten belül egyetlen szál végzi a trend mintavételezését. A példabeli 30 tábla is csak 1 trend, 1 adatgyűjtési és 1 közös alarmkezelő szálát hoz létre, amit a program megpróbál külön CPU magokhoz rendelni. S így a szokásos 4-8 CPU magot tartalmazó számítógépeket az X10 optimálisabban terheli.

Kérdés, hogy okoz-e ez a módosítás programtervezést érintő változást? Szerencsére nem igazán, mert a nagyobb projektekre is jellemző, hogy ugyanaz a ciklusidő az összes táblában. Ezért több szálla elosztva is pontosan ugyanaz a feladat hajtódott végre, mint a közösben. **Azt viszont érdemes megvizsgálni, hogy a változó táblák között nem szerepel-e véletlenül egy gyorsabb trend ciklusú, noha csak pár trend változó szerepel benne.** Mivel azokat is érdemes inkább a közös ciklushoz igazítani, nehogy lerontsák a rendszert.

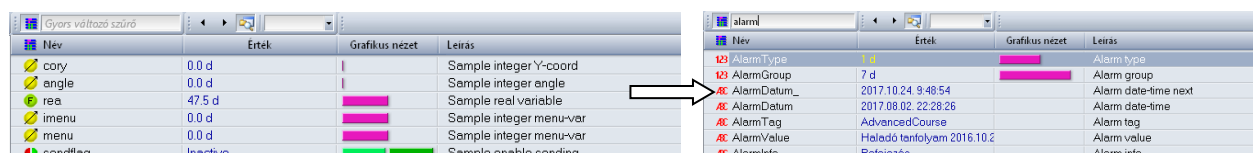
Hogy jobban érthető legyen, képzeljük el, hogy van egy projektünk, amiben 10 tábla van és mindegyik 1 perces alap ciklusidejű, a trendek száma pedig mondjuk 10,000. Van azonban egy szervíz tábla is, amiben csak 2db szervíz trend változó van 1 másodperces alapciklusidővel. Az X10 a fent elmondottak miatt 1 másodperces alapciklusra vált az egész alkalmazásra nézve. Nos ilyenkor indokolt azt megfontolni, hogy emiatt a 2db változó miatt tényleg 60-szorosára akarjuk-e gyorsítani a közös trend szálát. Az ui. a méréseink szerint kb. 15% felesleges terhelés növekedést okoz (hiszen 60-szor fogja megvizsgálni a program feleslegesen, hogy mintát kell-e venni). Ezért érdemesebb ilyenkor azt a két szervízváltozót is inkább percesre állítani.

3. Változók szűrése, listázása, tömbváltozók megjelenítése

A változók online megjelenítésére és kiválasztására szolgáló ablak is jelentős ráncfelvarráson esett át. Az egységesen balra rendezett értéklista felett a gyorskereső funkciója változtathatóvá vált:

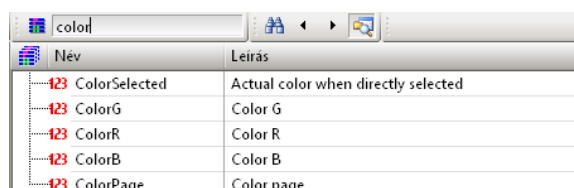


Ha megnyomjuk a bekeretezett gombot, a *gyors változó kereső* átvált *gyors változó szűrőre* és a begépett szövegtartalom szerint le is szűkíti a listát:



Igy lehet kiválasztani például az *Alarm*-mal kezdődő változó neveket.

Ugyanezek a lehetőségek szerkesztő üzemmódban is elérhetők és ugyanúgy működnek, vagyis a változók megjelenítése és kiválasztása egységes lett.

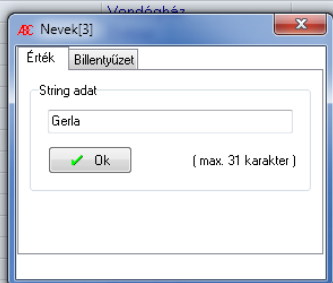


A korábbi módszerrel is lehet keresni (távcső ikon), a technológiai azonosítóra, leírásra vagy mértékegységre történő keresést a fent említettek kibővítik és interaktívvá teszik. Különösen nagyszámú változó esetén hasznos a név alapján való előszűrés.

Egyebekben a listát lehet rendezni bármelyik oszlopa szerint, strukturált változók esetén pedig nem bontja fel többé a <változónév>.attr típusú listákat, ami a színváltozó hozzárendelést áttekinthetőbbé teszi.

A fent felsoroltakon kívül van még egy nagyon hasznos módosítás: tömbváltozók megjelenítése. Az X9-ben mindig csak az első tömbelemet láttuk és csak azt tudtuk módosítani. Az X10-ben azonban a tömbváltozó elemeire bontható az ikon előtt megjelenő + gombbal:

Név	Érték	Grafikus nézet	Leírás
X6			String (message) variable
X7			String (message) variable
Fogy[0]	12344.0		Real variable
FogyMin[0]	10009.0		Real variable
FogyMax[0]	23468.0		Real variable
ScrollPos	57		Direct integer variable
Nevek[0]	Békés-Szarvasi út		String variable
Nevek[1]	Békés-Rákóczi út		String variable
Nevek[2]	Békés-Csatánkert		String variable
Nevek[3]	Gerla		String variable
Nevek[4]			String variable
Nevek[5]			String variable
Nevek[6]			String variable
Nevek[7]			String variable
Nevek[8]			String variable
Nevek[9]			String variable
Nevek[10]			String variable
Nevek[11]			String variable
Nevek[12]			String variable
Nevek[13]			String variable
Nevek[14]			String variable
Nevek[15]			String variable
Nevek[16]			String variable
Nevek[17]			String variable

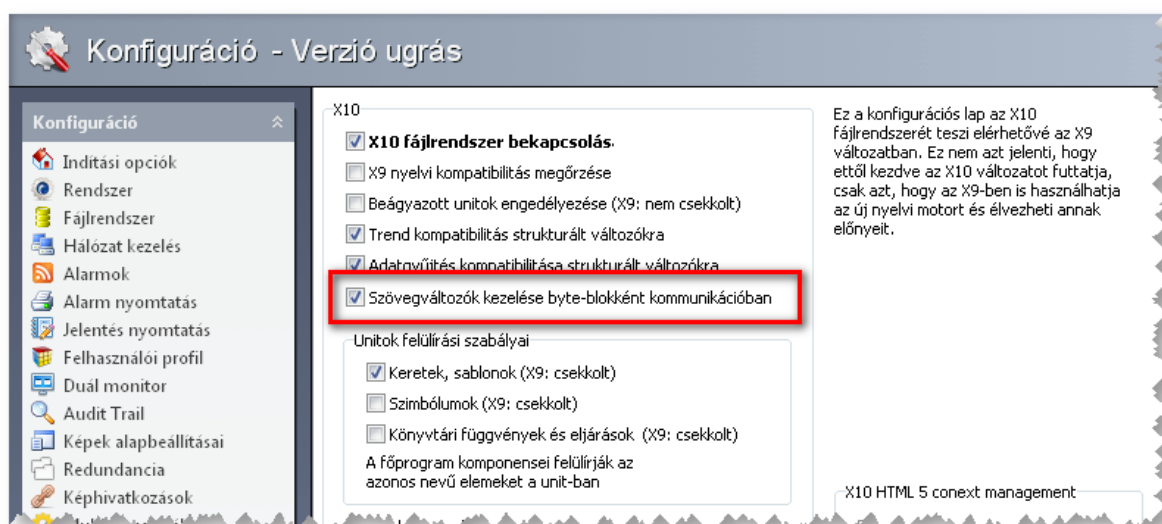


Mint látható, a tömbváltozó összes elemére rákattinthatunk és átírhatjuk az értékét.

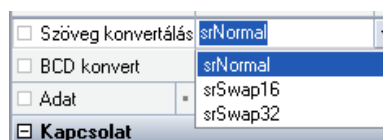
A képen eltérő színnel, szürkén megjelenő leírás mezők is új (*Real variable*). A változó jeltípusonként eltérő alapértelmezett leírását különböztetjük meg így a konfiguráció során általunk megadottól. Amennyiben a leírás szürke, akkor nincs konfigurált értéke. Egyébként fekete (kép teteje).

4. Szövegváltozók küldése és vétele byte-blokkban

Az X9 rendszerben a szövegváltozót mindig egészben lehetett adni, venni a kommunikációs üzenetekben, azaz fix 32- és 256 byte méretű blokkokban. Az esetek túlnyomó többségében azonban a szövegváltozó nem ilyen hosszú a PLC oldalon, hanem teljesen változó méretű, pl. 6 karakteres. Az X10-ben ezt egyszerűen a szövegváltozó mögé írt tömbszámossággként adhatjuk meg a kommunikációs üzenetekben. Például: STRINGVAR[6]. További különbség, hogy nem kell az üzenetben az adatnak a hosszal kezdődnie (mint az X9-ben), a szövegváltozót az első 0 karakter is terminálja (mint C-ben), de a hossza nem nagyobb mint a blokkhossz (példában = 6). Mindez egy konfigurációs paramétertől is függ:



Az így beolvasott, küldött szövegváltozó szó- és bytesorrend cseréjére is lehetőség van *Szöveg konvertálás* (*StringConvert*) tulajdonságon keresztül. Az *srSwap16* opció minden szomszédos byte-ot cserél, az *srSwap32* pedig az összes byte-ot 32-bites regiszterenként.



5. Verzió lekérdezése

Szükség lehet rá, hogy programból megkérdezzük, vajon X9 vagy X10 motor alatt futtatjuk-e az alkalmazást. Erre szolgál a

X10Version

parancs, aminek a visszatérő értéke attól függől true vagy false, hogy a rendszert melyik motor futtatja. A használatát az AdvancedCourse nevű demoalkalmazásban mutatjuk, ahogy a program a template-ben található emblémát cseréli az operációs rendszernek megfelelően.

6. X10 superprojekt

A superprojekt attól függően lesz X9 vagy X10 alapú, hogy magában a superprojektben ez be lett-e állítva. Könnyű elmenni mellette, ezért külön felhívjuk a figyelmet a superprojekt konfiguráció eme új checkboxára:

Önálló projektek listája:

Név	Projekt címe
Plctest	Plctest alkalmazás
Features	@Programszolgáltatások
Bio	Biodiesel demo
Provimi	PROVIMI KLIBA SA
Provicon	Provicon Web Pages
Clarkadam	Clark Ádám úszódarú
Szazviz	Százhalombatta ivóvízrend:
Advancedc...	Haladó tanfolyam 2017.11.22
Unittest	UnitTest alkalmazás

Hozzáadás ►

◀ Kivonás

Ürit

Kiválasztott projektek (super_projekt komponensek):

Név	Cím
Szazviz	Százhalombatta ivóvízrendszer...
Advancedc...	Haladó tanfolyam 2017.11.22

Főprojekt (elsőként betöltött projekt), kiválasztás a fenti listából jobb egérgomb

Advance Haladó tanfolyam 2017.11.22

Projekt neve: **pampam1 alkalmazás**

☒ X10 Superproject

A rendszer ui. nem tud különbséget tenni X9 és X10 között projektenként, amikor együtt futtatja őket. Vagy mind X10 vagy mind X9. Ettől a checkboxtól függően, és függetlenül attól, hogy egyébként projektenként mire konfiguráltuk az alkalmazásokat külön-külön. Ha utólag kívánjuk megváltoztatni ezt, akkor a superprojekt VPR fájljába szövegesen kell beírni:

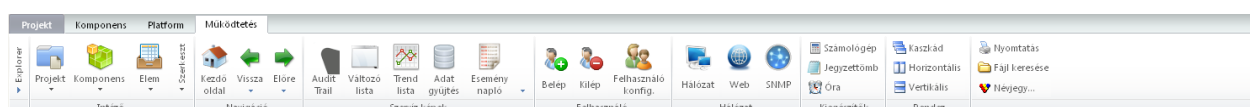
```
Project begin
  Title= "RG_EK alkalmazás"
  Icon= "bin\icon\squares.bmp"
  CommandLine= "RG_Main, RG_Energia, RG_Tablá, RG_Gozfogado, RG_Hocser, RG_Hokozp, RG_Szny1, RG_Szny2, RG_Kond, RG_Gkazan"
  X10FileSystem=Yes
End
```

Bevezetés a VISION X10-be

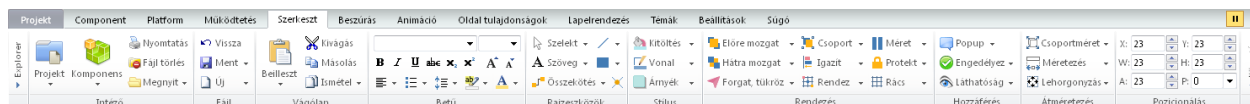
Ismét hatalmas fejlesztésen vagyunk túl, elkészült az ipari folyamatmegjelenítés eddigi leghatékonyabb eszköze, a VISION X10. A fejlesztés alapját a közel három évtizedes alkalmazói tapasztalat, a felhasználói visszajelzések, a számítógépek hardware és software fejlesztésének legújabb tendenciái, valamint a számítástechnikai programozói rendszerek fejlődése határozta meg. A program minden tekintetben a legkorszerűbb megoldásokhoz igazodik ismét.

Mindent kicseréltünk. Teljesen új, sokkal felhasználó barátabb lett a program felhasználói felülete, új a fordító, új a rajzoló, de újak még a rendszer belső implementációjának az elemei is. A teljes menürendszer is kicserélődött és egyszerűsödött:

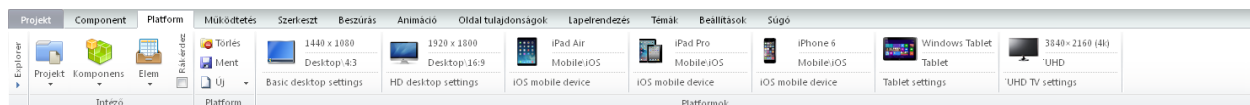
Működtetés:



Szerkesztő ribbon:



Platform válogató:



Az eredmény egy nagyságrendileg gyorsabb megjelenítő rendszer. A program eddig is a leggyorsabbak közé tartozott, de a mostani verzió verhetetlen. Miközben az új programot sokkal könnyebb kezelni, és minden tekintetben többet is tud. Igyekszem azt is rögzíteni, mindjárt az elején, nehogy a VISION felhasználó megrettenjen: a program megőrizte kompatibilitását a korábbi verziókkal és a kezelés (szerkesztő pl.) is visszaállítható ugyanarra a technikára. Nézzük a részleteket.

1. Ablak kezelés

A rendszer átalakítását egészen az alapoknál kezdtük, mindjárt magának a Windows-nak az ablak-kezelőjénél. A Windows alaphelyzetben az egyes ablakok tartalmát, az ún. *window context*-et másolásolja a képernyőre, bitmap mozgatással (*BitBlt*). Ennek a megoldásnak azonban sok hátránya van. Többek között az, hogy több megjelenített elem esetén azokat először egyetlen windows bitmap-ben kell összemásolni (renderelni), ami több ezer objektum esetén eléggé fáradságos és időigényes művelet, másrészt ha nagyítani kell, az objektumokat külön-külön kell felnagyítani, majd megint összerakni. Nem megoldott továbbá, hogy több szinten (layeren) lehessen objektumokat rajzolni.

1.1. Window rendering

A Windows context renderelője helyére saját megoldás került, ami kapásból kétszer gyorsabb, mint elődje, de van sok további alapvető különbség is. Az itt szereplő sebesség adatok egy 3,4 GHz sebességű i7-3770 CPU-val és Nvidia GTX 650-es grafikus kártyával szerelt gépen mértek.

Sebesség növekedés összetevői:

Rendszer elem	Korábbi	Új	Sebesség növekedés
1. Window context redrawing Full HD kép megjelenítése	5 ms	2 ms	2,5 - szeres

1.2. Vektorgrafika alap szinten

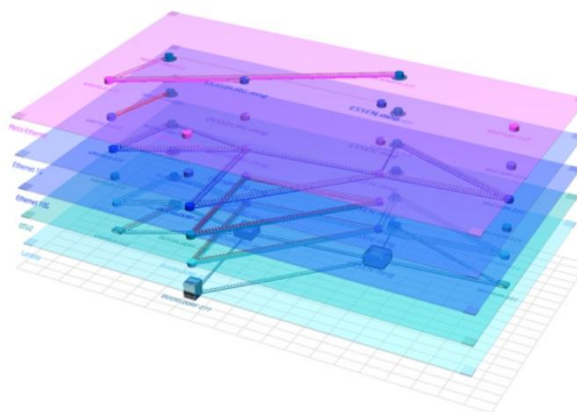
A rajz a layeren nem bitmap többé, hanem egy vektor sík, amiből a Window tartalma a vektorgrafikából ismert *supersampling* eljárással készül. **Ezáltal az objektumok tetszőlegesen nagyíthatók anélkül, hogy bármit újra kellene rajzolni**, ill. helyette az alap szinten implementált ablak megjelenítő végzi ezt a műveletet, ami így sokkal gyorsabb, de főleg sokkal kényelmesebb:

Rendszer elem	Korábbi	Új	Sebesség növekedés
2. Vektorgrafika	egyedileg kellett az objektumokat újrajzolni	Supersampling	kb. 3 - szoros

Ez a technika akkor mutatja meg a hatását, ha egy ablakba belenagyítunk. Eddig ez úgy ment, hogy átméreteztük az ablakot, és ha a kép ablakmérethez igazított volt, akkor az egér elengedésekor – némi várakozás után – megjelent az új, nagyított kép, miután mindent újrajzoltunk. **Mostantól ez folytonos, mivel nem kell semmit újrajzolni**, tehát nem is kell annyit várni. Az ablak tartalmát ezentúl magában az ablakban is felnagyíthatjuk a görgetővel, és szerkesztés közben is választhatunk vektoros és pixelgrafikus nagyítás között. Hovatovább a nagyítás animált, vagyis eléggé látványos.

1.3. Multilayer technológia

A harmadik fontos különbség, hogy mindez egyszerre több síkon is lehetséges, amelyek **teljesen önálló pozícióval, átlátszósággal és nagyítással rendelkeznek!** Ez talán a **legfontosabb és leghatásosabb különbség**. Az új ablak kezelő a layerek egymásra másolásával jeleníti meg a képet, vagyis a síkok rendeléséről nem a VISION-nek kell gondoskodnia, következésképp annak nincs processzoridő igénye sem. Ellenben ezt a műveletet maga az ablak kezelő végzi, ami megint sokkal hatékonyabb:



Rendszer elem	Korábbi	Új	Sebesség növekedés
3. Többosztú rederek	egydi objektum összmásolás	multi-layer technológia	változatlan kép esetén kb. 2 - szeres; de: ügyes layer-struktúrával akár 10-szeres is lehet

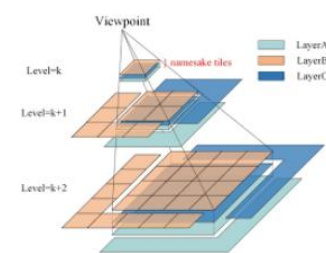
Egyszerű gondolatsorral belátható, miért sokkal gyorsabb ez az eljárás összességében. Képzeli el, hogy csak egy animációt frissítünk a képen, ami önálló layeren van, az egész bonyolult kép pedig egy

másikon. Ezen a szinten a frissítés sebessége csak ettől az egyetlen objektumtól függ, hisz rajta kívül nincs semmi más, nem ütközik semmivel, miatta nem kell semmit újrarajzolni. Most a változást az új ablak kezelőnek kell összemásolnia a másik layer tartalmával. Tehát mintha csak átadtuk volna a munkát egy további szintnek – első ránézésre úgy tűnik, nem spóroltunk meg semmit. Igenám, de a renderelés csak abban négyszögben szükséges, amiben az objektum van; adott esetben csupán 64x64 pixelt érint a művelet ahelyett, hogy az egész full-HD képet újrarajzolnánk.

Ennél azonban fontosabb, hogy a rendszernek sokkal több szabadságot biztosít. Eddig ugyebár csak két szintünk volt: egy statikus és egy dinamikus. Mostantól akár 32 különböző szintünk is lehet, teljesen eltérő dinamikával, ciklus idővel. A layerek egymástól függetlenül nagyíthatók, mozgathatók, átlátszóvá tehetők vagy eltüntethetők feltételesen.

Visszatérve ahhoz a problémához, hogyan nagyítsuk egy ablak tartalmát, mindenek előtt azt kell megértenünk, hogy egy ablaknak nem minden elemét kell nagyítanunk általában és szinte sohasem egyformán. Például a fejlécben található menüsört nem, de egy táblázatban lévő szöveget sem kell, csupán a táblázatot magát. Aztán vannak kötések jobbra, balra, le, fel, ami megint azt eredményezi, hogy az **egyes objektumokat teljesen eltérő módon kell hozzáigazítani a változó méretű ablaktartalomhoz**. Na pontosan ezért nem lehetett “normálisan” megoldani a vektorgrafikus képek nagyítását a korábbi verzióban, amikor is csak egy síkunk volt, mert vagy mindig azonos pixelméretűek voltak az objektumaink és kötöttek az ablakszélekhez, vagy változó méretűek, azaz nagyíthatóak, de akkor nagyított a program mindent, a fejlécben található menüt is és az összes szöveget.

A multilayer technika változást hoz ezen a téren. A menüket, ribbon kontrol sort, sőt az egész template-et is felrajzolhatjuk egy külön layer-re, amit nem nagyítunk (csak az ablakszélhez kötünk). Egy másik része a képnek közben nagyítható; egy hamadik, egy táblázat mondjuk, csak átméreteződik stb. Ráadásul a nagyítás lehet arányos, központosított stb.



A multilayer technológia sok előnnyel jár és sok új lehetőséget teremt, miközben minden sokkal gyorsabb is lett, mint korábban.

2. Windows rajzelemek cseréje

Megint az a kérdés vetődhet fel az olvasóban, hogy erre mi szükség volt? Valójában nagyon sok oka van. Például az előző fejezetben ismertetett vektorgrafikus rajzolást sem tudtuk volna megoldani, de probléma volt az is, hogy a Windows nem elég gyors. Ez desktop környezetben nem annyira zavaró (a felhasználó nem veszi észre, ha fél másodperc alatt rajzolunk fel egy képet), de Web-es környezetben már problémát okozott, ha egyszerre mondjuk 20-an léptek rá a szerverre (20x0.5sec!). Ráadásul a Windows saját rajzelemei nem támogatják a multithreading-et, így még az a lehetőség is elmaradt, hogy a rajzolást több szálon futtatva gyorsítsuk. Nem véletlenül használ saját rajzoló rutinokat a legtöbb web browser is; a Webkit alapú rendszerek még a betűket is maguk festik, ahogy mi is.

A képelemek rajzolása a VISION X10 változattól kezdve sokkal gyorsabb, futtatható több szálon, s persze támogatja a supersampling-et, ami a vektorgrafikához kell.

2.1. Szövegek kirajzolása

Akármilyen durván hangzik, mostantól a TrueType fontok minden egyes pixelét VISION rutinnal rajzoljuk fel – kihagyva a Windowst és vele az idióta Clear-Type-ot. Mint a legtöbb Web browser! Nem volt egyszerű feladat, mivel a fontrajolás rendkívül sok megfontolást igényel – korrekt nagyíthatóság, antialiasing, stílusok, lumineszcenciától való függőségek stb. –, de végül megoldottuk. S ha már hozzányúltunk, kibővítettük a fontok stíluskészletét is pár hasznos új típussal. Ilyen a hullámmal való

aláhúzás hibás állapot megjelenítésére, a gradiens színátmenet és a karakterek, sorok távolságának az állíthatósága (mint a CorelDraw-ban). De a legfontosabb eredmény ez:

Rendszer elem	Korábbi	Új	Sebesség növekedés
4. Betű rajzoló	Windows	VISION	20 - szoros!

Ez annyira hatásos változtatásnak bizonyult, hogy a VISION X9.64 változatába is betettük. A leglátványosabban a változó szerkesztőben lehet látni a sebesség növekedést, amikor a strukturált változók 50 oszlopos listáját görgetjük: nincs semmilyen megakadás, akárhogy rángatjuk is az egeret, pedig százával kell szélsimított truetype szövegeket köpní a képre.

2.2. PNG rajzoló

Sajnos a PNG rajzoló is elég gyenge volt, ami a sebességet illeti. Ez persze nem véletlen, hisz egyrészt a PNG fájlok tömörítettek, ezért felrajzolás előtt ki kell bontani őket, másrészt a PNG alpha layerét össze kell renderelni a képháttérrel. S akkor a képet még csak nem is nagyítottuk. Az új rajzoló pedig:

Rendszer elem	Korábbi	Új	Sebesség növekedés
5. PNG rajzoló	Embarcadero	Vision	23 - szoros!

A sebesség növekedés fő oka az, hogy az alpha rendering a Windows context renderelőjébe lett beépítve. Remélem, ezen a ponton kezdi élvezni a kedves olvasó ezt az amúgy megjelehetősen száraz, a néhány szektás rendszerprogramozón kívül mást nem érdeklő témát! ☺

2.3. Bmp, Jpg, Gif...

Ezekből is a leghatékonyabb megoldások kerültek a programba. Például a Jpeg-re maga az Intel gyártotta az eljárást a saját processzorára optimalizálva. Állítólag ennél nincs jobb.

A legfontosabb változás azonban, hogy az összes bitmap korrektül nagyítható és kicsinyíthető ezentúl. A korábbi, tisztán pixeles nagyítás helyébe komoly *pixelprocessing* lép. Például a nagyítás az ún. *bidirectional interpolation* nevű eljárással valósul meg kontraszt korrekcióval, a kicsinyítés pedig a leglátványosabb eredményt biztosító, már említett *supersampling*gel. Ezáltal még az MS-Word képnagyítójánál is szebb eredményt kapunk, pedig a Word-be épített megoldás egyike a legjobbaknak a világon.

Minden bitmap forgatható, perspektívába állítható, lehet áttetsző stb. Ezek egy része eddig is létezett, lehetett például a bitmapet átlátszóvá tenni, de csak Windows Bmp-t. Mostantól nincs korlát és még a gradiens transzparencia is lehet bármilyen szögű.

Végül, de nem utolsó sorban bekerült a színkorrekció lehetősége a programba (világosság, kontraszt, színdinamika), és lehet azonos átlagos tónust is forszírozni. Hol van erre szükség? Manapság egyre gyakrabban rakunk *flat* ikonokat a képre és egyszínűeket, hogy ne a szinkavalkád vonja el a felhasználó figyelmét (ld. *flat design*). Ezért fekete fehér konverziót kérünk a programtól. Majd észrevevesszük, hogy a korábban színes ikonjaink a feketétől az egészen világos szürkéig változtatják a tónusukat. Az említett opcióval azonban ez mostantól kiegyenlíthető.

3. Real-time megjelenítés

Hatalmas jelentőségű változás, hogy az új ablakkezelési szolgáltatásoknak köszönhetően az ablakok tartalma real-time, valós időben jelenik meg ezentúl. Mindez ráadásul **külön szálon fut!** Bármilyen bonyolult képet csináljunk is, nem fékezi a program többi részét, mivel külön CPU magot terhel, és azt akár 70%-ra is kiterhelhetjük anélkül, hogy a program kezelhetőségét ez jelentősen befolyásolná!

Mivel továbbá a VISION X10 képek jóval gyorsabbak is, mint a korábbi változatban, **a képek bonyolultsága és megjelenítési sebessége akár egy nagyságrendet is változhat.** A képek ciklusideje valódi 10ms-ra redukálható alapból is – tehát a real-time processzálas bekapcsolása nélkül –, s így végül akár 100 változást is megmutat a program másodpercenként. Pontosan 100-at (100 Hz).

Háttér: A real-time ablakkezelés nem triviális, nem véletlenül nem foglalkozik ezzel a specialisták szűk körén kívül senki a világon. Kapásból gond az, hogy az ablaktartalmat a Windows is festi, amint azon ablakok vonulnak át, vagy monitoron kívüli területekre mozgatjuk és onnan vissza. Ezeknek az újrefestési műveleteknek szinkronizálnak kell lenniük a külön százból indított és a fent leírtak miatt eléggé szapora rajzolási műveletekkel, a mozaikra bontott képtartalom megváltoztatását pedig egymással is szinkronizálni kell. Most pedig tegyük ehhez hozzá, hogy a kép görgethető, nagyítható és eleve több layer-ből is áll...

Ismereteink szerint nincs ilyen real-time megjelenítésre képes más megjelenítő rendszer jelenleg a világon. Olyan gyors real-time megjelenítő, mint a VISION X10 pedig biztosan nincs!

4. Fordító

Teljesen új, sokkal gyorsabb fordítóprogram került a VISION-be, ami képes a korábbi XSDL képleírók feldolgozására, de persze sok új lehetőséget is rejt. A hatékonyabb eljárás rendkívüli eredménnyel járt:

Megszűntek az objekt fájlok!

A program ezentúl közvetlenül a forrásból dolgozik, mint a *Javascript* alapú rendszerek is például. A belső fordító JIT (Just-In-Time) alapon működik, vagyis amikor egy új képre váltunk, beolvassuk a szövegfájlt és már nézhetjük is a képet. Egy átlagos kép esetén a feldolgozási idő 5 ms (a korábbi kb. 200ms-os linkelési időhöz képest!).

A fordítóra eddig azért volt szükség, mert az objekt fájlok linkelése jóval gyorsabb volt, mintha mindig mindent újrarendeltünk volna. Mivel azonban ez a probléma megszűnt, hisz a források fordítása baromi gyors lett, objekt fájlokra sincs szükség többé! Mit jelent ez a *baromi* gyors?

Az egyik legnagyobb projektünk 75,000 változóból álló változó listájának a lefordítása mindössze 850ms, vagyis kevesebb, mint 1 másodperc az új rendszerben. A korábbi változatban ez 3 perc volt, ami durván 200-szoros sebesség növekedés! Az igazsághoz tartozik, hogy ez nem általános, mivel a változófájl fordítása korábban a mérettel négyzetesen nőtt, az új rendszerben azonban közel lineárisan. Vagyis kisebb var fájlok esetén az eltérés is kisebb, de mindenképpen igen jelentős, átlagosan kb. 50-szeres. Erre ugyanakkor szükségünk is van, hisz az objekt fájl megszűnt, s mivel a lefordított var fájl a régi rendszer is viszonylag gyorsan betöltötte, most ezzel a gyorsabb szöveg alapú JIT fordítónak kell versenyeznie.

A képfájlok és a többi taszk fordítása is sokkal gyorsabb lett, de ott nem olyan látványos az eltérés, mint a var fájlok esetén. Viszont még így is sokkal gyorsabb még a régi OBJ betöltésénél is linkelésénél is.

Az összes forrásfájlt figyelembe véve a fordító átlagos sebességnövekedése:

Rendszer elem	Korábbi	Új	Sebesség növekedés
6. Fordító	Vision X9	Vision X10	15 – szörös!

5. Futtatás

Ami a kód futtatását illeti, a legnagyobb változás a képek objekt listájának a kiértékelésénél van, itt minimum 100-szoros a sebességnövekedés, azonban a többi rendszer objektum esetén (ciklikus program, alarm stb.) csak kb. 2,5-szeres. A végrehajtható kód sebessége régi rendszerben is nagyon gyors volt, de még ezt is sikerült túlszárnyalni:

Rendszer elem	Korábbi	Új	Sebesség növekedés
7. Futtató	Vision X9	Vision X10	2,5 - szeres!

A sebességnövekedés összetevőit foglalja össze a következő táblázat:

Rendszer elem		Régi	Új	Sebesség növekedés	Hátasosság
1. Ablak megjelenítő	5 ms	2 ms	2,5 szeres	★★★★★	
2. Vektorgrafika	egyedileg kellett az objektumokat újrarajzolni	Supersampling	~ 3 szoros	★★★★★	
3. Többszintű rederezés	egyedi objektum összmásolás	multi-layer technológia	változatlan kép esetén kb. 2 szeres	★★★★★	
4. Betű rajzoló	Windows	VISION X10	23 szoros	★★★★★	
5. PNG rajzoló	Embarcadero	VISION X10	20 szoros	★★★★★	
6. Bitmap rajzoló	Embarcadero	VISION X10	8 szoros	★★★★★	
7. Fordító	VISION X9	VISION X10	15 szörös	★★★★★	
Összesített	Vegyes	VISION X10	~ 15 szörös	★★★★★	

A hatásosság csak egy arányszám a különböző összetevők viszonyára, hogy ezek az egész működő rendszeren belül mekkora jelentőséggel, milyen hatással bírnak.

6. Nyelv

Az új nyelvi motor nem gyorsabb csupán, de rengeteg új nyelvi elemmel is bővült és megszűntek a korábbi korlátok például az indexelésnél. Lehetőség van továbbá a nyelv szintaxisának a megválasztására. Jelenleg a megszokott és a korábbi változatokkal kompatibilitást biztosító XSDL mellett Pascal-ban és C-ben programozható a rendszer, de fejlesztés alatt áll a Java, ill. Javascript és az Xcode. Nemsokára mindenki olyan nyelven programozhatja a VISION-t, ami a legjobban fekszik neki. Nézzük a részleteket.

6.1. Új XSDL nyelvi elemek

Nyelvi elem	Leírás, példa
Case utasítás	Megjelenít a <i>case</i> utasítás (C-ben <i>switch</i>): <pre>case idx of 1: szivattyu1=1; Szivattyu2=2; 4: szivattyu1=2; 9: szivattyu1=3; end</pre> Az érték nem csak szám lehet, de szövegkonstans is! Amellett, hogy egyszerűbb elágazni a programban így, a <i>case</i> utasítás jóval gyorsabb is, mivel csak az értéknek megfelelő feltételes ág értékelődik ki a végrehajtás során. Break nem kell az utasítások végére (mint a C-ben).
Konstans tömb	<pre>const2[8]:=[101,102,103,104,105,106,107,108];</pre> Felsorolhatjuk tehát egyszerűen a tömbelemeket.
Változó felsorolás	Lehetővé vált a változó megadásánál a felsorolás, azaz egyszerre több változót is megadhatunk ugyanarra a típusra: <pre>i[10],j,k,l: integer;</pre> Ez a megadási mód a VISION változófájlban is megengedett: <pre>Kond_1, Kond_2: 300,133920,20,Kond,m,1,0,0,6,0,6,0,0,0,8065 'Kondenz</pre>
Direkt típusmegadás	A változófájlban ezentúl hivatkozhatunk a típusra közvetlenül is, vagyis nem kell többé a változókat típus szerint csoportokba rendezni: <pre>Integertrends Kond_1: 300,133920,20,Kond,m,1,0,0,6,0,6,0,0,0,8065 'Kondenz</pre> Helyett írható ez is: <pre>Kond_1: Tintegertrend 300,133920,20,Kond,m,1,0,0,6,0,6,0,0,0,8065</pre> Ennek a megadási módnak az a fő előnye, hogy a változók sorrendjét teljesen függetleníti a változók típusától.
Variant	Megjelenít a <i>variant</i> típus: <pre>v: Variant;</pre> Ennek az az érdekessége, hogy a típusa az értékadásnál dől el (<i>v=false</i>; <i>v=123</i>; <i>v="szöveg"</i>). Tartalmazhat bool, integer, real, string változót, sőt változó objektumot is!
Univerzális Xnumerics, Xdigits, Xstrings	Az <i>Xnumerics</i>, <i>Xdigits</i>, <i>Xstrings</i> típusok ezentúl használhatók függvények, eljárások argumentumában is, vagyis a cím szerinti univerzális változó-átadás teljesen általánosan használhatóvá vált (korábban csak szimbólumok és képek

	argumentumai között lehetett szerepeltetni, eljárásokban, függvényekben nem).
Változó-átadás bármilyen mélységben	Nincs többé korlát a paraméter átadást illetően szimbólumok és képek egymásba ágyazásakor, mivel valódi cím szerinti átadás történik. Bármilyen mélységben is ugyanúgy hivatkozhatunk tehát a változóra és annak összes elemére (Min, Max, Dim, Mean...).
Break	Megjelent a <i>break</i> utasítás, amivel ciklusokból lehet kiugrani: <code>break;</code> Egyaránt használható <i>for</i> , <i>repeat-until</i> és <i>do-while</i> ciklusokban.
Nem kötelező az eljárás-paraméter	Eljárásoknál és függvényeknél ezentúl nem kell kötelező argumentum (korábban legalább 1 paramétert kellett szerepeltetni a function/procedure/ kulcsszó után): <code>datum:: string function begin datum=Date; end</code>
Strukturált változók kiterjesztése	Az X9-ben a strukturált változóknak csak az első tagját lehetett átadni a Tstructure típusú paraméternek. Az X10-ben bármelyik részét: Például a <code>Bpest.IVker.fiok1.gozsziv</code> változónak egyaránt áthadó a <code>Bpest</code> , a <code>Bpest.IVker</code> , vagy a <code>Bpest.IVker.fiok1</code> tagja, míg korábban csak a <code>Bpest</code> -et lehetett struktúra prefixnek tekinteni. Az X10 valódi szöveg behelyettesítéssel operál.

Emellett a belső string műveletek mind unicode-ban történnek (bármilyen nyelv) és a szöveghossz sem korlátozott 255 karakterre, mint korábban. Egy-egy közbülső szöveg lehet akár 2^{31} karakter hosszúságú is. Persze a stringváltozó mérete továbbra is korlátozza, hogy mennyi tárolható el ebből, amikor értékül adjuk azt egy globális változónak.

Az X10 eljárások és függvények realtime környezetben is megkötés nélkül használhatók, mivel bármelyik szálból újrahívhatók (reentráns). Ez korábban csak azoknál az eljárásoknál volt lehetséges, amelyek nem használtak lokális változókat az eljárástömbön belül. Ezek ui. a taszk szintjén globális változók voltak, így pl. ugyanazt a ciklusszámlálót használta volna két különböző szálból hívott eljárás. Ez a megkötés megszűnt, mivel minden paraméter és lokális változó is a *stack*-ben van.

6.2. Új változó típusok

Van pár hasznos új változó típus. Ezek a következők:

Változó típus	Leírás
BCD	Analóg, azaz skálázható BCD szó (OMRON)
BCDtrend	Analóg trend, azaz skálázható és trendelhető BCD szó (OMRON)
Extended	10-bájtos lebegőpontos szám extra pontosságú műveletekhez

6.3. Új objektum típusok

Van pár új objektum típus is:

Objektum típus	Leírás
TStringList	Klasszikus stringlist szövegfájlok programból való feldolgozására
TCSV	Cella szerkezetű objektum CSV és XLS fájlok adatainak cellánkénti feldolgozására (csv: TCSV; csv.Cells[1,2]=123;)
TDataSet	SQL utasítások eredményhalmazainak a hatékonyabb feldolgozására szolgál (ds: TDataSet; ds=sql "DATA.Table","select * from Table"; ds.Field("Adat").AsString...; ds.Next...)

Ezeket az objektumokat is pont úgy hozhatjuk létre, mint a képi objektumokat. Például:

```
Csv: Tcsv;
```

A megszüntetéséről nem kell külön gondoskodni (ahogy a képi objektumok megszüntetéséről sem).

A hagyományos objektumokhoz képest az a nagy különbség, hogy ezekben nem csak tulajdonságokat, de metódusokat és függvényeket is megtalálhatunk. Például:

```
csv.LoadFromFile("c:\Doksik\Adatok.csv");
```

További részletek a kibővített teljes dokumentációban.

Gurka Tibor
vezető fejlesztő