Magyar Mérnöki Kamara
Kiadványsorozata

126.

Az NMHH ESZTER aktuális verziójának használata során feltárt hiányosságok, problémák kezelése az AutoCAD programozási eszköztárával

Hírközlési és Informatikai Tagozat

FAP-2024/107-BMK (HIT)

Magyar Mérnöki Kamara
Kiadványsorozata 126.

Az NMHH ESZTER aktuális verziójának használata során feltárt hiányosságok, problémák kezelése az AutoCAD programozási eszköztárával

MMK FAP azonosító:
2024/107-BMK (HIT)

Budapest, 2024. október

A sorozat szerkesztője:

WAGNER ERNŐ
a Magyar Mérnöki Kamara elnöke

Készült a Magyar Mérnöki Kamara Hírközlési és Informatikai Tagozatának gondozásában, a 2024. évi Feladat Alapú Pályázatok pénzügyi keretéből.

A kiadvány a Magyar Mérnöki Kamara tulajdona. Másolása, teljes terjedelmében való közzététele csak a Kamara engedélyével lehetséges. Minden jog fenntartva.

Szerzők:

Egri Sándor

Lektorálta:

Lukács Tamás

Témafelelős:

Petruska István

Kiadó:

Magyar Mérnöki Kamara
1118 Budapest, Budaörsi út 125/A

fap@mmk.hu, www.mmk.hu

TARTALOMJEGYZÉK

1. Vezetői összefoglaló

A távközlési hálózat tervezési terület évek óta problémákkal küzdött, majd az NMHH a Hír-Közmű rendszerének bevezetésével 2023. áprilisától kötelezővé tette az EHO XML alapú engedélyezési eljárásrendet.

Az XML állomány előállítására akkor az egyetlen eszközként az NMHH megrendelésére készült, AutoCAD Map 3D alapú ESZTER tervezéstámogató rendszer állt rendelkezésre. A tervezők a rákövetkező hónapokban szembesültek a problémával, és a helyzet súlyosságával.

Az újonnan fejlesztett ESZTER program rengeteg hibával, koncepciószintű problémákkal került a tervezőtársadalom részére átadásra, valamint a program erőforrás igénye és az AutoCAD Map 3D alap szükségessége hardver és szoftver beruházásra kényszerítette a tervezőket.

A helyzetet súlyosbította, hogy nem álltak rendelkezésre az XML alapú tervezéshez a szolgáltatók hálózatának adatai, így a tervezőket a tervezett hálózati objektumok XML formába történő átdolgozásán felül a meglévő hálózati elemek konverziójának többletmunkája is terhelte.

Bár maga az ESZTER program azóta új verzióval rendelkezik, mely javított a korábbi súlyos hibákon, a rendszeren még mindig rengeteg javítani és fejleszteni való van.

A tervezők helyzete azóta bonyolódott azzal, hogy a szolgáltatók elkezdték feltölteni hálózati adatszolgáltatásukat a Hír-Közmű rendszer adatbázisába, de onnan adatot kinyerni nem lehet, tekintettel, hogy az erre szolgáló NMHH-s modul még nem készült el.

A Magyar Mérnöki Kamara, illetve annak Hírközlési és Informatikai Tagozata támogatásával és segítségével a tavalyi évben elkészült az ESZTER hatékonyabb használatát bemutató segédlet.

A rendszer még hosszú időre informatikai feladatot fog jelenteni a tervezők számára, de jelen segédlet segítséget nyújthat abban, hogy az XML és ESZTER alapú tervezés folyamatainak automatizálására, gyorsítására megfelelő segédprogramokat készít(tet)hessenek a tervező kollégák.

2. Bevezető

Az távközlési területen (de más szakágak területén is) jellemző, hogy az elmúlt inkább évtizedek során a különböző tervező vállalkozásoknál, cégeknél, a nekik dolgozó műszaki rajzolóknál rengeteg saját stílus alakult ki a tervek dokumentációs, de főképp rajzi állományainak előállítására.

Külön blokk és fólia rendszert alakítottak ki, melyekkel való munkát sokszor saját illetve külső fejlesztésű programok, AutoCAD kiegészítők segítették.

Ezen sablonok és eszközök a Hírközlési Felügyelet illetve későbbi jogutódja, a Nemzeti Hírközlési Hatóság (mely 2010.-es jogszabályi hatáskörbővítés után már Nemzeti Média- és Hírközlési Hatóságként funkcionál) építési, használatbavételi, stb. engedélyezési eljárásainak megfelelő tervdokumentációk előállítására teljesen alkalmasak voltak. Az akkori közmű- és szakhatósági engedélyezési folyamatok miatt az engedélyezés nagyformátumú másológépeken másoltatott tervekkel történt, néha rendkívül nagy (30-40) példányszámmal, így a tervek kizárólag fekete-fehér (illetve másológéptől függően más színnel monokromatikusak) voltak.

Ez idő tájt kezdtek megjelenni a különféle távközlési szolgáltatók igényei szerint kidolgozott saját sablonrendszerei illetve nyilvántartási rögzítéshez alkalmazott AutoCAD beépülő moduljai. Ezek a tervezők és rajzolók számára kötöttségként jelentkeztek a saját módszereikhez képest.

A régi időktől AutoCAD programot (akár még a régi DOS alapú R13 vagy R14 verziótól) használó tervezők és rajzolók munkamódszere, „workflow”-ja nem a nyilvántartó programokba kódolt „kattints a gombra-tedd le az objektumot-kattints újra” ismétlődő monoton és lassú folyamatával történt, hanem már meglévő saját elemek másolásával, parancsrövidítések alkalmazásával, saját kiegészítők általi gyors módosításokkal jóval hatékonyabban készítették el a szükséges rajzi állományokat.

Az eltelt évek alatt ilyen sablon és nyilvántartó rendszerek jöttek-mentek, távközlési cégek olvadtak össze, de a GPON optikai előfizetői hálózatok elterjedése megkövetelte új dokumentálási és nyilvántartási rendszerek bevezetését.

Eközben előbb a 324/2013. (VIII. 29.) Kormányrendelet alapján bevezetésre került a Lechner Tudásközpont által menedzselt E-közmű rendszer. Ezen rendszeren a közműadatok egységes formátumú biztosítására, továbbá a folyamatok egységes felületen, jogszabályi háttérrel támogatott határidőkkel operálva történő közmű-üzemeltetői engedélyezéshez nyújtott lehetőséget, míg a Nemzeti Média- és Hírközlési Hatóság (NMHH) új engedélyezési folyamatai lehetővé, később kötelezővé tették, hogy az engedélyezési dokumentáció PDF formátumban kerüljön a hatósághoz, illetve a

bevonni szükséges szakhatósági szervezetekhez. A PDF lehetőséget biztosított arra, hogy a tervek már színkódolva tartalmazzák a különféle fontos objektumokat, akár külön kapcsolható fóliákon, összességében a papír alapú példányoknál hatékonyabb felhasználással. Természetesen a kivitelezéshez továbbra is szükség volt papír alapú tervpéldányokra, de a fentiek miatt lecsökkent a példányszám igény és a nagyformátumú színes nyomtatás, másolás költségének csökkenése miatt ez már sokszor szintén színes volt.

Viszont a szolgáltatói sablonok legtöbbször ezt teljesen lehetetlenné tették (lásd egyik nagy szolgáltató tiszta piros, óriási blokk feliratokkal dolgozó nyilvántartási rendszerbe bedolgozáshoz szükséges sablonját), így kialakult egy kettős tervezési-rajzolási módszer, melynek során a saját sablonokkal és kiegészítőkkel elkészül egy kiviteli terv, olyan formában, hogy azt a hatóságok és a kivitelezők kezelni és értelmezni tudják, majd ez kerül újra rögzítésre az érintett nyilvántartó rendszerekbe.

Ezzel el is érkeztünk a jelenbe, az NMHH új Hír-Közmű rendszere élessé válása, az annak alapját képező Egységes Hírközlési Objektummodell (EHO) és ennek előállítását segítő Egységes Szakági Tervezéstámogató Rendszer 2023. április 3. óta kötelező jelleggel történő bevezetéséig.

Az EHO egy XML alapú modell, a távközlési hálózatok, azok objektumainak és a közöttük lévő kapcsolatok leírására, azaz **nyilvántartására** szolgál, így a tervezőknek ismét egy egyáltalán nem flexibilis rendszerrel kell dolgozni. Ez leginkább kiváltási tervek készítésénél jelentkezik, mely a nyomvonal jellegű objektumok módosulásával jár.

Az EHO XML állomány előállításához az NMHH fejlesztetett egy AutoCAD Map 3D kiegészítőt, mely az Egységes Szakági Tervezéstámogató Rendszer (ESZTER) nevet kapta. Sajnos ez merőben eltérő módszert használ a hálózati objektumok tárolására, nem a hagyományos AutoCAD elemeket, mint vonal(lánc) és blokk, hanem úgynevezett szakági adatmodellt és a hozzá kapcsolódó megjelenítési modellt. Előbbi egy adatbázis alapú rendszer, azaz a rajzba bekerülő objektumok valójában nem AutoCAD rajzelemek, nem módosíthatóak és másolhatóak azzal a korábbi eszköztárral, mely éveken, évtizedeken át szolgálták a tervezőket, rajzolókat.

Ezen felül azon tervező vállalkozások, akik eddig jól elboldogultak és továbbra is elboldogulnának egy korábban megvásárolt sima AutoCAD verzióval vagy akár valamelyik alternatívával (ZWCAD, BricsCAD, stb.), azok ezután az ESZTER verzióigénye miatt már csak előfizetéses rendszerben működő legújabb verziójú AutoCAD Map 3D bérlésére kényszerültek, nem beszélve annak szakági adat- és megjelenítési modelljének erős erőforrás igénye miatt szükséges hardver fejlesztésről.

A szolgáltatók által fejlesztetett saját, XML előállítására használatos kiegészítők, nem ilyen szakági adatmodellel, hanem az AutoCAD Map 3D objektumadat (object data) eszközrendszerét használják, mely a legalább – ha korlátozottan is – lehetőséget ad a megszokott AutoCAD workflow alkalmazására.

(Megjegyzendő, hogy technikailag lehetséges akár régi, sima AutoCAD verzióban is plusz adatok tárolása a DWG rajzokban.)

A teljesen más munkamódszert igénylő új rendszer használatán és elfogadásán tovább rontott, hogy az ESZTER koncepciószintű, program és adatbázis hibákkal került kiadásra. A bevezetés óta eltelt időben az NMHH javítási és új funkciókkal bővítési fejlesztési csomagot rendelt meg a fejlesztőtől, így 2024. májusában az új 1.4.0 EHO verzió életbe lépésével együtt az új ESZTER verzió is publikálásra került.

Jelen segédletnek nem célja és nem is lehet célja, hogy ezeket a koncepcióbeli és még fennmaradt hibákat kijavítsa, tekintettel, hogy az ESZTER forráskódjával a Hatóság nem rendelkezik, így abban javítani nem lehet. Így marad a különböző programozási eszköztárakkal történő funkcióbővítés, esetleges feldolgozás gyorsítás, illetve az adatmodell adatainak saját módszer szerinti kezelése.

3. Távközlési hálózati objektumok adatmodellezése

Mielőtt rátérnénk a segédlet konkrét témájára, mint minden programozási feladat megoldása előtt tudni kell milyen adatokat és azokat milyen formában és kontextusban kell kezelni, így a következő fejezet a távközlési hálózatok adatmodellezésésről, illetve konkrét esetként a EHO XML és az ESZTER adatmodelljeiről szól.

Hogy hogyan lehet a távközlési hálózatok objektumait, azok tulajdonságait modellbe foglalni? Egymillió és egy féle módon, ahány tervezőt, programozót, egyéb szakembert megkérdezünk, mindenki tudna egy új modellt összeállítani.

Hogyan lehet egy *hatékony és jól kezelhető* modellben leírni? Ez már egy jóval összetettebb kérdés.

Természetesen a válasz a *feladatnak megfelelő legegyszerűbb* módon, de hogy ezt hogy lehet megvalósítani, ahhoz szükséges ismerni a felhasználás eseteit (*use-case*), illetve hogy milyen adatokat, paramétereket kell ezekhez tárolni.

3.1. Az EHO, mint adatmodell

Az EHO XML létrehozására az NMHH Hír-Közmű projektjéhez volt szüksége. A hatóság Hír-Közmű weboldalán [1] található a projekt célja „**egységes, országos hírközlési infrastruktúra nyilvántartása**”.

A hálózati objektumok (és kapcsolataik) az Országos Hírközlési Adatbázisban (OHA) kerülnek tároásra. Az OHA adatcsere formátuma maga az EHO, melyet a szolgáltatók adatszolgáltatás benyújtása illetve a tervezési engedélyezési eljárások során szükséges használni.

A nyilvántartási rendszerek célja egyértelműen statikus objektumok tárolására irányul, például egy könyvtári rendszer, ahol érkeznek új könyvek, esetleg idővel selejtezésre kerülnek, de azok, mint objektumok nem változnak.

Ellenben a távközlési hálózatok egyáltalán nem statikusak. Miközben egy új lakópark kialakítása során megtervezett új GPON hálózat jó eséllyel évekig, évtizedekig nem fog már változni, addig az ország teljes területére vetítve ez már egyáltalán nem teljesül. A távközlési tervezők nem csak a szolgáltatók megbízásából új hálózatok tervezése a feladatuk, az ország infrastrukturális fejlesztése kapcsán rengeteg helyen szükséges kiváltás tervezése, amely a meglévő hálózati objektumok módosulásával jár.

Míg az előző könyvtáras példában a könyvek leselejtezése (és a kikölcsönzés során történő esetleges elvesztése) analóg azzal, mikor egy kiváltás során elbontásra

kerülnek a hálózati elemek, de rengeteg esetben ettől jóval összetettebb forgatókönyvek fordulnak elő:

- Egy alépítmény szakasznak csak egy részét kell elbontani, új megszakító szekrény ráépítéssel. Ekkor a megmaradó alépítményi csövek mint, mint objektumok a valóságban tulajdonképpen ugyanazok, csak a nyomvonal és így a hossz változik.

  Az OHA-ban viszont nincs lehetőség megrövidítésre, átalakításra. A tervezőnek ilyen esetben a teljes korábbi alépítményt bontani kell a benyújtandó EHO XML állományban, továbbá a megmaradó alépítmény szakaszokat is újra létre kell hoznia, ugyanazokkal a paraméterekkel (típus, átmérő, tulajdonos, stb.), mely lényegesen több plusz munka a korábbi tervezési eljárási folyamathoz képest.
- Fenti eset, kiegészül azzal, hogy az alépítményben üzemelő kábelekre az új megszakítókban új kötések kerülnek.

  Ekkor a tervezőnek már a nem csak az alépítményi csöveket, hanem a kábeleket a következő kötésig (**!!!**) kell újra létrehozni, mely már mondhatjuk exponenciálisan több feladatot és időt jelent!
- Új kerékpárút építése miatt az amellett haladó oszlopsor merevítéseit (huzalkötél, támfa) elbontják, de a meglévő egygyámos oszlopok helyben duplagyámossá építik, és elforgatják. Ekkor az oszlop és az egyik gyám meglévő, megmaradó, csak már elforgatás paraméterrel.

  Ugyanez a helyzet, mint fent, hogy hiába nem kellene a tervezőnek az oszlopsor kábeleivel foglalkoznia, mégis a bontás-újra létrehozás folyamatát itt is el kell végeznie.
- Kerékpárút vagy járdaburkolat építése miatt a megszakító szekrények födémjét szintbe kell helyezni, a fedlapok cseréjével (pl. térkővel burkolható típusúra). Ekkor a megszakító ugyanaz, csak a térbeli kiterjedése (magasság) és a fedlap típusa változik.

  A fenti esetek extrém példája. Habár csak egy megszakító érintett, ki tudja hány cső, béléscső és kábel objektummal kell foglalkozni!

Jelen segédlet készítése idején a fenti problémák még azzal bonyolódnak, hogy az NMHH Hír-Közmű rendszerbe bekerült adatok lekérdezési felülete még nem készült el, magyarán nem lehet a meglévő hálózatokról adatot, információt kapni.

Tetézi ezt az, hogy a Magyar Telekom a hálózati nyilvántartási adatait (nagyon sokszor hibás geometriai adatokkal és egyéb paraméterekkel) elkezdte az OHA-ba feltölteni, és mivel onnan a tervező nem kaphat adatszolgáltatást, sokszor csak az engedélyezési

eljárás során szembesül a tervező az engedélyezésre benyújtott EHO XML adatai és az OHA adatai közötti ütközéssel.

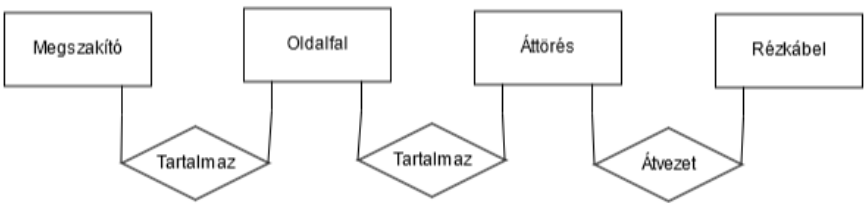
A fenti koncepcionális problémákon felül maga az EHO XML felépítése is hagy kivetni valót maga után.

Magának az XML leíró nyelv specifikációjakor egy ember és gép által is megérthető formátum létrehozása volt a cél, de valljuk be nem lehet teljesen komplett az a felhasználó, aki egy többezer objektumból álló GPON hálózat EHO XML állományát olvasgatja (hacsak nem kénytelen pár módosítást eszközölni a Hír-Közmű fenti illetve az ESZTER később tárgyalandó hibái miatt).

Az XML koncepciójából eredő terjedelmességet jóval össze szedettebb, minimálisabb objektumtípus számmal lehetett volna kompenzálni, valamint a tulajdonságok XML tagok között történő leírása helyett lehetett volna paramétereket is használni.

Továbbá, bár az az elgondolás, hogy a távközlési objektumok között bizonyos kapcsolati reláció legyen, az alapvetően jó, de a megvalósítás ebben az esetben sem az egyszerűség irányába mutat.

Tekintsünk például egy megszakító szekrényt és egy abban (annak falán) végződő alépítményi csövet.
Az EHO-ban ez jelenleg így modellezhető:



1. ábra Rézkábel és megszakító kapcsolatlánca az EHO-ban

A fenti modell EHO XML-ben így néz ki (az egyes objektumok tulajdonságainak mellőzésével):

```
<objektumok>
  <megszakító>
    <azonosító>megszakito_001</azonosító>
  </megszakító>
  <megszakító_oldal>
    <azonosító>oldalfal_001</azonosító>
  </megszakító_oldal>
  <áttörés>
    <azonosító>attores_001</azonosító>
  </áttörés>
  <rézkábel>
    <azonosító>rezkabel_001</azonosító>
  </rézkábel>
</objektumok>
<kapcsolatok>
  <tartalmaz>
    <kiinduló_objektum>megszakito_001</kiinduló_objektum>
    <vég_objektum>oldalfal_001</vég_objektum>
  </tartalmaz>
  <tartalmaz>
    <kiinduló_objektum>oldalfal_001</kiinduló_objektum>
    <vég_objektum>attores_001</vég_objektum>
  </tartalmaz>
  <átvezet>
    <kiinduló_objektum>attores_001</kiinduló_objektum>
    <vég_objektum>rezkabel_001</vég_objektum>
  </átvezet>
</kapcsolatok>
```

1. kódrészlet Megszakító és Rézkábel objektum kapcsolatlánca EHO XML-ben

Valljuk be, mennyivel egyszerűbb és hatékonyabb lenne az alábbi modell:

```
<megszakító azonosító=”megszakito_001”>
</megszakító>
<rézkábel azonosító=”rezkabel_001”>
  <átvezet>
    <vég_objektum azonosító=”megszakító_001” oldalfal=1 pozíció_x=40 pozíció_y=30>
    <vég_objektum azonosító=”megszakító_001” oldalfal=3 pozíció_x=50 pozíció_y=60>
    <vég_objektum azonosító=”megszakító_002” oldalfal=1 pozíció_x=40 pozíció_y=30>
  </átvezet>
</rézkábel>
```

2. kódrészlet Megszakító és Rézkábel objektum kapcsolat paraméterekkel

Látható, hogy paraméterekkel, és a kapcsolatok részleges beágyazásával sokkal jobban átlátható lehetne az EHO XML, különösen, hogy az a jelenlegi változatban az

objektumok XML tagjai és azok kapcsolat tagjai között több száz, ezer sor távolság is lehet.

Azután ott vannak az egyes objektumok tulajdonságainak értékkészletei. Senki nem hivatkozik egy ADSS optikai légkábelre úgy, hogy „*Fényvezető egymódusú önhordó fémmentes mikro légkábel ADSS*", vagy egy Qvr kábelre „*Emelt védőtényezős, vazelinnel töltött, műanyag szigetelésű*" néven. Nyugodtan lehetne ezeket rövidebben leírni, maximum az EHO kézikönyvben egy mellékletben részletezni.

Továbbá ott voltak az 1.3-as EHO-ban az értelmezhetetlen tulajdonságok, úgymint cső kapacitás vagy az optikai kábelek kötött és vágott szálai. (Az 1.4-es EHO kézikönyve legalább már magyarázatot ad ezekre, de annak kiadásáig a tervezők csak fogták a fejüket.)

3.2. Az ESZTER adatmodellje

Ha már az EHO sem túl jó koncepció, mit mondhatunk az ESZTER-re?

Attól még, hogy a Hír-Közmű térinformatikai alapú rendszer, nem kellett volna a felhasználói oldalon is ilyen megoldást erőltetni. Az EHO XML előállításhoz szükséges adatok tárolását – mint azt a Magyar Telekom iTTM rendszere is mutatja – meg lehet oldani Map 3D objektumadatokkal (*object data*) is, de még akár sima AutoCAD *Xdata* és *Dictionary/Xrecords* adatstruktúrákkal is, ha a fejlesztő nem vágná túl nagy fába a fejszéjét, és az XML előállítás minimális feladatain túl nem akarna „tervezéstámogató" rendszert előállítani.

Az AutoCAD Map 3D szakági modellen alapuló rendszer annyira különbözik a sima AutoCAD objektum alapú tervezéstől, mintha valakinek az eddigi autóval megtett külváros-belváros ingázása helyett repülővel kellene ugyanezt megtennie. Nemcsak hogy egy másabb, jóval összetettebb rendszert szükséges megtanulni, de jócskán erőforrás igényesebb is.

Az ESZTER jelenlegi kiadásban több mint 400 **jellemzőosztály** (adatbázis tábla) van. Ez egy komplett nagyvállalat-irányítási rendszer komplexitásával ér fel, nem csoda, hogy az első kiadásban rengeteg hiba adatbázis eredetű hiba volt, illetve az objektumok törlése nem volt lehetséges (nem volt szabad), mivel a szükséges összetett eseményvezérlők nem voltak megírva.

Bár az 1.4-es EHO-hoz készített új kiadás javított hibákat, és pár hasznosabb új funkcióval is bővült, még mindig van az ESZTER rendszeren fejleszteni és javítani való.

Az akkori és még mindig tapasztalható hibák, illetve koncepciószintű problémák tárgyalása különálló kötet méretét érné el.

Mivel a program forráskódjával és annak felhasználási jogával kizárólag a fejlesztő Fornax Zrt. rendelkezik, külső fejlesztő nem tud a funkcionális részben (a különböző paletták és mögöttes kódjaiban) fejlesztést eszközölni, kizárólag az szakági adatmodell adatbázis vagy API (programozási interfész) szinten.

Bár láttuk, hogy a tervek EHO XML állományainak előállítása során rendszerszintű problémákkal állunk szemben, azaz tekinthetjük ezt egy szélmalomharcnak, mégis nézzük meg milyen lehetőségek állnak rendelkezésre az AutoCAD Map 3D programfejlesztési eszköztáraiban a problémák kezelésére.

Az C# nyelv, illetve az AutoCAD fejlesztőeszköztára és a .NET keretrendszer összetettsége túlmutat jelen segédlet keretein. A bemutatandó példák hasznosak az elinduláshoz, „*de csak az ajtót mutatom meg, azután neked kell belépni rajta*”.

4. Az AutoCAD programozási eszköztárai

Tekintettel, hogy jelen segédlet nem programtervező informatikusoknak, hanem mérnök, kimondottan távközlési tervező mérnök kollégák részére készült, így a lehető legtöbb szemléletes kép és a legkevesebb számítástudományi kifejezés alkalmazásával került megírásra, de sajnos vannak elkerülhetetlen szakszavak, melyek nélkül nehéz a programozás koncepcióját megérteni.

Az viszont elvárható, hogy a felhasználó képben legyen a memória, processzor, fájl/állomány, futás/futtatás, stb. alapvető fogalmakkal, illetve programokat, számítógépes eszközöket tudja használni.

Ezen kívül hasznos, ha minimális szinten érti az olvasó az angol szavak jelentését, tekintettel, hogy a programozási nyelvek és fejlesztőeszköztárak kulcs szavai és függvénynevei általában angol nyelven alapulnak.

A következő történeti áttekintés olyan kifejezéseket tartalmaz, melynek megértése nem szükségeltetik a segédletben tárgyaltak elsajátításához, de szükségesek az AutoCAD programozási eszköztárának teljesebb bemutatásához.

A segédlet fő feladata a szakági adatmodell rendszerek jellemzőosztályainak (adatbázis tábláinak) kezelését segítő programkód bemutatása.

Ezen példakódok segítségével az érdeklődő mérnök kollégák szakáguk problémái és feladatai megoldására tudnak majd saját segédprogramokat, AutoCAD kiegészítőket létrehozni.

4.1. Az AutoCAD programozási eszközeinek fejlődése

Az Autodesk fejlesztői az AutoCAD programot már kezdeti időszakban a bővíthetőséget szem előtt tartva fejlesztették.

Alkalmazásprogramozási felületként (API, *Application Programming Interface*) a korai verziókban bevezetésre került az *AutoLISP* nyelv, illetve a C nyelv alapú ADS (AutoCAD Development System).

Az AutoLISP a későbbi verziókban is megmaradt, sőt jelentős funkcióbővítésen esett át, míg az ADS-t felváltotta az új, C++ alapú *ObjectARX* (AutoCAD Runtime eXtension) programozási felület, majd később beépítésre került Visual Basic for Application makrók futtatására alkalmas motor, és az COM Component Object Modell, mellyel gyakorlatilag bármely nyelvből lehetővé vált az AutoCAD megfelelő komponensei funkcióinak hívása. Mivel az ObjectARX használatához szükséges C++ nyelv elsajátítása sok időt vesz igénybe, sok fejlesztő a könnyen megtanulható Visual Basic 6 fejlesztőkörnyezettel és a COM hívások használatával készített kiegészítőt.

A Microsoft 2002-ben vezette a .NET keretrendszert, a C# és VB.NET programnyelveket, melyekkel egy rendkívül nagy alap függvénykönyvtár, két viszonylag könnyen tanulható nyelv állt a fejlesztők rendelkezésére, több más hasznos funkcióval (pl. ún. szemétgyűjtő [garbage collector] memóriakezelés, típusbiztonság, nyelv interkompatibilitás).

Ahogy a .NET kezdett egyre népszerűbbé válni, úgy az Autodesk fejlesztői is elkezdték az ObjectARX programozási felület .NET környezetben használható változatának elkészítését.

Sajnos az elmúlt pár években az Autodesk fejlesztői dokumentációi és oktatóanyagai terén is minőségi és mennyiségi romlás történt, míg a fő eszközökre (főképp az alap AutoCAD-re) vonatkozóak legalább frissítve vannak az időszakos API módosítások után, addig a kevésbé használtak (mint például a számunkra fontos szakági modell) esetén rendkívül szűkösek az információk.

Jelen segédlet elkészítésében is régi, néha évtizedes blogbejegyzések, sok esetben az Archive.org Internetes archívum által elmentett, évek óta nem elérhető weboldalak és tutorialok segítettek.

5. Programozási alapozás

Ahhoz, hogy megértsük a programozás koncepcióját, az alábbi hat fogalmat kell megérteni: változó, függvény, kifejezés, elágazás, ciklus és kódblokk. Előbbiek mérnöki tanulmányok kapcsán matematikai tárgyakból ismerősek lehetnek.

1. **Változó**: A változó általunk meghatározott absztrakt eszköz, mely egy adott értékhalmazból tetszőleges értéket felvehet. A valóságban nem más, mint a számítógép memóriájában tárolt valamilyen adatra hivatkozás, ahol, mint a neve mutatja, az adat a program futása során változhat. akár külső okokból (pl. fájl beolvasása) vagy mi magunk akarjuk változtatni (pl. a lenti ciklus feltétel változójának módosítására).

2. **Függvény**: Akárcsak a mérnöki tanulmányok matematikai kurzusain, a függvény bemenő változó(k) értékei alapján ad valamilyen kimenő értéket. A program függvényei segítségével tudjuk a változók értékeit a nekünk szükséges módon változtatni, egy futtatandó programkód részlet, melynek van egy neve és paraméterei. A teljes program futtatása során ezzel a névvel hivatkozunk erre a kódrészre.

3. **Kifejezés**: változók, függvények és a nyelv által biztosított műveletek (más néven operátorok, gyakorlatilag, mint a matematikai műveletek), tetszőlegesen összetett kombinációja. Amikor a program futása egy kifejezéshez ér, az abban lévő függvények meghívásra kerülnek, majd az összes visszatérési érték és a változók értékei az kifejezésben lévő műveletekkel meghatározásra, „kiszámításra” kerülnek.

4. **Elágazás**: Pontosan, mint a valós életben, elérünk egy útelágazáshoz, megnézzük az útjelző táblákat és attól függően megyünk valamelyik irányba. Ugyanígy egy programban az elágazásnál valamilyen változó értékét vizsgálva megyünk egyik vagy másik irányba, azaz kerül egyik vagy másik programkód részlet futtatásra.

5. **Ciklus**: Minden általunk végzett ismétlődő tevékenységnél ciklust végzünk: például súroljuk az odaégett lábost, de csak óvatosan, ne sértsük meg a zománcot, megnézzük, van-e még rajta folt, ha igen, megint dörzsölünk rajta, megint megnézzük, és folytatjuk, míg el nem tűnik a folt. Ugyanígy a programban a ciklus addig futtat egy programkód részletet, míg egy meghatározott változó értéke (a feltétel) igaz (tiszta-e a lábos) hamis (koszos-e még) nem lesz.

6. **Kódblokk**: Az előző öt alapfogalom tetszőleges egymás utáni sorozata, mely a legtöbb nyelvben külön jelekkel vagy kódszavakkal kerül határolásra, például a C alapú nyelvekben kapcsos zárójelekkel, LISP esetén sima zárójelekkel vagy Pascal nyelvben `BEGIN` és `END` kulcsszavakkal.

5.1. Változók típusai

Nem mehetünk el amellett a fontos kérdés mellett, hogy milyen lehet a változó értéke. A fenti matematikai példában nem mindegy, hogy természetes számokkal (pozitív egész számok) vagy valós számokkal (tizedes törtek) foglalkozunk. Programok és programnyelvek esetén az alábbi *számunkra fontos* típusok vannak.

1. **Egész**: nemigen kell nagyobb magyarázat, pozitív vagy negatív egész számok és a nulla lehet. Általában számlálásra, valós dolgok megszámlálására használatos.

2. **Valós**: ezt sem kell részletesen tárgyalni, gyakorlatilag a számegyenes bármely számáról lehet szó (a valóságban a programnyelv és a processzor típusától függ), mi majd a rajzok elemeinek koordinátái és hasonlók tárolására használjuk majd.

3. **Sztring**: szöveg, tetszőleges karaktersorozat, melybe nyelvtől függően tartoznak különböző karaktertáblák

4. **Mutató**: tulajdonképpen a számítógép memóriájának egy meghatározott egységének kezdőcíme. Rendkívül fontos a hardver közeli programozási nyelvekben (C, C++), de sem a segédletben tárgyalandó AutoLISP, sem a C# nyelv tárgyalásakor nem találkozunk majd ilyen típusú változóval (habár maga a mutató a LISP alapjának belső működéséhez és a C# egyes adatszerkezeteihez elengedhetetlen).

5. **Összetett**: A fentiek kombinálása az nyelv adta nyelvi eszközök segítségével.

Természetesen ez egy igen erős egyszerűsítés, ettől sokkal több alap (vagy akár kevesebb) típus lehet a programozási nyelvekben, és azoknak altípusai is lehetnek (például előjeles vagy előjel nélküli számok), de későbbi fejezetekben az AutoLISP és C# tárgyalásánál visszatérünk a kérdésre.

5.2. Függvények és egy hasonlók

Egy program tulajdonképpen függvények láncolata, mely során egymásnak különböző változókat adnak át (*meghív* egy másik függvényt) és azok feladatuk befejeztével *adnak vissza* valamilyen értékkel.

A fenti egyszerűsítés a programozási nyelvi függvényről persze sokkal összetettebb a valóságban, és persze nagyban függ az adott programozási nyelvtől, de általánosságban a következő koncepció szerint épülnek fel (kimondottan az általunk használni kívánt C# nyelv szemszögéből):

```
public int DeleteFeatures(MgResourceIdentifier fsId, string className,
string filter)
{
MgFeatureCommandCollection cmds = new MgFeatureCommandCollection();

MgDeleteFeatures delete = new MgDeleteFeatures(className, filter);

    cmds.Add(delete);

MgFeatureService fs =
          this.Connection.CreateService(MgServiceType.FeatureService);

MgPropertyCollection result = fs.UpdateFeatures(fsId, cmds, false);

var ip = result.GetItem(0) as MgInt32Property;

if (ip == null){
      return -1;}
else {
      return ip.GetValue();}
}
```

3. kódrészlet

A fenti C# nyelvű példa lehet elrettentő, de nem kell ahhoz érteni, hogy lássuk a színkódolás alapján a koncepciót.

Minden **változó** `piros`, minden **típus** `lila`, minden **függvény** `világoskék` és minden vastag **`kék`** szó az adott programozási nyelvben **kulcsszó**.

A fenti `DeleteFeatures` függvény egy `int` (azaz egész) típust ad vissza, a bemenő változói `fsId`, `className` és `filter`, melyek rendre `MgResourceIdentifier`, `string` és `string` típusúak (előbbi egy összetett, utóbbiak pedig a már tárgyalt sztring típusok).

A { } kapcsos zárójelek közötti kódblokk tulajdonképpen a függvény magja, ez kerül minden egyes alkalommal futtatásra, ahányszor a `DeleteFeatures` függvényt a program más része *meghívja*.

A kódblokkban az egyenlőségjelek nem az általunk megszokott matematikai egyenlőség, hanem a nyelv eszköze arra, hogy a jobb oldalon lévő kifejezés (itt függvényhívás visszatérési) értékét a bal oldalon lévő, lila típusú piros nevű változóban eltároljuk.

A függvény a feladatainak elvégzése után (ez lehet akár programkód közepe is!) a `return` kulcsszó utáni kifejezés értékét adja vissza az őt hívó programrésznek.

5.3. Elágazás és ciklus, azaz vezérlőszerkezetek

Elágazások

Már a fenti kódban is látható, hogy egyszerű eseteken is szükség lehet a program lineáris futásáról eltérni, az lenti kódrészlet egy elágazás (IF) szerkezetet mutat.

Az IF, azaz magyarul HA kulcsszó után zárójelben lévő kifejezés igaz értéke esetén az első kódblokk (az első kapcsos zárójelpár közötti rész), míg hamis értéke esetén, ha van ELSE ág, az ott lévő kódblokk fut le, míg az ELSE ág hiánya esetén hamis feltétel esetén csak átlép az IF ág kódblokk utáni következő kifejezésre.

```
IF (logikai feltétel)
{
      //kódblokk a feltétel igaz értéke esetén
}
ELSE
{
      //kódblokk a feltétel hamis értéke esetén
}
```

4. kódrészlet IF/ELSE elágazás (feltételes utasítás) általános szerkezet

Egyes nyelvekben van lehetőség az elágazások egymásba láncolására, illetve speciális nyelvi eszközök állnak rendelkezésre a többszörös értékű feltételvizsgálat esetére is.

```
SZAM := BEVITEL( „Adj meg egy számot” )
IF (SZAM < 10 )
{
      KIIR( „A szám kisebb mint 10” )
}
ELSE
{
      KIIR( „A szám kisebb mint 10” )
}
```

5. kódrészlet IF/ELSE elágazás példa

Ciklusok

A programban egyes kódrészletek többször történő futtatására szolgáltnak a ciklusok. A ciklus addig hajtódik végre, amíg a megadott feltétel igaz.

```
WHILE (feltétel)
{
      //futtatandó kódblokk a feltétel igaz értéke esetén
}
```

6. kódrészlet WHILE ciklus

A feltétel a kódblokkban egy meghatározott változó adott értékűvé válásának vizsgálatát jelenti, mely a gyakorlatban legtöbbször vagy külső tényezőtől függő esetet (pl. a felhasználó adott válaszát) vagy adott számosságot elérő esetet (pl. egy 0 alapértékű változót minden ciklus futtatáskor 1-el növelve, az kisebb-e mint pl. 10, azaz így a ciklust tízszer futtatjuk le).

```
SZAM := 0
WHILE (SZAM < 10 )
{
      KIIR( „A szám még kisebb mint 10, értéke: ” , SZAM )
      SZAM := SZAM + 1
}
```

7. kódrészlet WHILE ciklus, számlálás példa

Ezen rövid elméleti alapozás után nézzük a projekthez használandó C# nyelv és a kapcsolódó .NET keretrendszer alapjait.

6. A C# programozási nyelv és a .NET keretrendszer, valamint az ObjectARX.NET alapjai

Habár logikusabbnak tűnhetne előbb tárgyalni az LISP nyelvet, mint a jóval összetettebb C# nyelvet és a .NET keretrendszert, az AutoCAD Map 3D nem nyújt eszközt a szakági adatmodell (adatbázis) AutoLISP nyelven keresztül történő kezelésére, így saját eszköztárat szükséges előállítani, melyhez az ObjectARX programozási felület .NET változata kerül felhasználásra.

6.1. C# nyelv és a .NET keretrendszer

Az C# nyelv úgynevezett objektum-orientált, imperatív nyelvként kerültek megalkotásra. Előbbi azt jelenti, hogy a programkóddal a megoldandó feladat területén lévő objektumok, absztrakt fogalmakat különálló, de egymással kommunikáló programrészekbe modellezik le, míg utóbbi pedig, hogy ezen egységeken belül a program egymást követő műveletek lépéseiként kerül meghatározásra.

A fejezet csak nagyon csekély részét tudja bemutatni a C# nyelvnek és a .NET keretrendszernek, az ezen túlmutató részek esetén a tisztelt olvasóra marad, hogy ismereteit bővítse az irodalomjegyzékben szereplő oktatóanyagok segítségével.

6.1.1. A C# rövid története

A C# nyelvet a Microsoft fejlesztette ki a 2000-es évek elején, hogy egy korszerű, objektumorientált programozási nyelvet hozzon létre, amely versenyre kelhet más népszerű nyelvekkel, mint például a Java és a C++. A C# első kiadása 2002-ben jelent meg, a Microsoft .NET keretrendszer 1.0 verziójával együtt.

A C# nyelv kifejlesztése szorosan összefügg a Microsoft .NET keretrendszerének létrejöttével. A .NET keretrendszer egy hatalmas gyűjteménye a programozási könyvtáraknak és eszközöknek, amelyek megkönnyítik a programok fejlesztését és futtatását. A keretrendszer célja az volt, hogy egységes programozási környezetet biztosítson, amelyben a különböző platformokon és technológiákon (pl. asztali alkalmazások, webes alkalmazások, mobilalkalmazások) írt programok könnyedén együttműködhetnek.

A .NET keretrendszer egyik legnagyobb előnye a nyelvfüggetlensége. Ez azt jelenti, hogy több programozási nyelv is használhatja a keretrendszer szolgáltatásait, köztük a C#, a Visual Basic és az F#. A C# azonban a .NET elsődleges nyelve, és a keretrendszer számos funkciója kifejezetten a C# fejlesztőinek készült.

Az évek során a .NET keretrendszer folyamatosan fejlődött, és ma már több platformon is elérhető, beleértve a Linuxot és a macOS-t is. Ennek köszönhetően a C# programozók már nem korlátozódnak a Windows operációs rendszerre, hanem szinte bármilyen platformra fejleszthetnek alkalmazásokat.

6.1.2. A C# alapfogalmai

Az 5. fejezetben tárgyalt alapokre építve nézzük a C# nyelvet eszközeit.

6.1.2.1. Adattípusok

A C# nyelvben két lényeged típust kell elkülöníteni, az **érték** típusokat és a **referencia** típusokat. Az érték típusok azok amelyek az adatok tényleges értékét tárolják. Ezek a típusok a memória ún. stack területén tárolódnak, ami gyorsabb hozzáférést biztosít számukra.

Az alapvető **érték** típusok közé tartoznak:

- *int* : Egész számok tárolására szolgál, és -2,147,483,648 és 2,147,483,647 közötti értékeket vehet fel.

```
int szam = 42;
```

- *double* : Valós számok tárolására szolgál, és nagyobb pontosságot kínál, mint az int. A double típus 15-16 számjegyig precíz.

```
double tizedesSzam = 3.14;
```

- *char* : Egy karakter tárolására szolgál. A char típust egyetlen karaktert tartalmazó egyszeres idézőjelek közé kell zárni.

```
char betu = 'A';
```

- *bool* : Logikai értékek tárolására szolgál, amely csak két értéket vehet fel: true (igaz) vagy false (hamis).

```
bool logikaiErtek = true;
```

A **referencia** típusok nem tárolják az adatokat közvetlenül, hanem az adatokra mutató hivatkozásokat tárolnak. Ezek általában a memória ún. heap területén helyezkednek el, és a Garbage Collector („szemétgyűjtő”, a lefoglalt, de már nem használt memóriarészek felszabadításáért felelős rendszermodul) kezeli őket.

Az alapvető *referencia* típusok közé tartoznak:

- *string* : Karakterláncok tárolására szolgál. A string típus immutable, ami azt jelenti, hogy miután létrehoztuk, nem változtathatjuk meg az értékét, módosításkor új példány kerül létrehozásra, a régit pedig a garbage collector szabadítja majd fel.

- *object* : Az összes típus ősosztálya, amely lehetővé teszi bármilyen típusú adat tárolását. Az object típus bármilyen típusú objektumot képvisel.
- *tömbök* (arrays): A tömbök több azonos típusú érték tárolására szolgálnak. A tömbök fix méretűek, ami azt jelenti, hogy a létrehozáskor meg kell határoznunk a méretüket, és az alapvetően nem változtatható.

6.1.2.2. Vezérlési szerkezetek

A C# nyelvben a program futása a különböző vezérlési szerkezetek segítségével valósul meg, amelyek lehetővé teszik az elágazásokat és az ismétléseket. Az alábbiakban részletesen bemutatjuk a C# vezérlési struktúráit.

Elágazok

- *if-else* szerkezet: Az if utasítás segítségével elágazásokat hozhatunk létre, amelyek lehetővé teszik, hogy a program különböző utakat kövessen a logikai (igaz/hamis) feltételek teljesülése alapján.
  Az if szerkezet általános formája:

```
int szam = 10;
if (szam > 0)
{
    Console.WriteLine( "A szám pozitív." );
}
else if (szam < 0)
{
    Console.WriteLine( "A szám negatív." );
}
else
{
    Console.WriteLine( "A szám nulla." );
}
```

8. kódrészlet A C# nyelv if/else szerkezete

- *switch* szerkezet: A switch szerkezet több lehetséges program útvonal kezelésére alkalmas, amikor egy adott többértékű változó értékét vizsgáljuk.
  A switch szerkezet általános formája:

```
char betu = 'A';
switch (betu)
{
    case 'A':
        Console.WriteLine( "A betű az 'A' betű." );
        break;
    case 'B':
        Console.WriteLine( "A betű az 'B' betű." );
        break;
    default:
        Console.WriteLine( "Más betű." );
        break;
}
```

9. kódrészlet A C# nyelv switch szerkezete

Ciklusok

- *while* szerkezet: Megegyezik az alapfogalmak részben tárgyalttal.

```
int szam = 0;
while (szam < 5)
{
    Console.WriteLine(szam);
    i++; // növeljük eggyel az i értékét (inkrementálás)
}
```

10. kódrészlet A C# nyelv while szerkezete

- *for* szerkezet: A for ciklus kifejezetten hasznos, amikor tudjuk, hogy hányszor szeretnénk végrehajtani a ciklust. Általános formája:

```
for (int szam = 0; szam < 5; szam++)
{
    Console.WriteLine(szam);
}
```

11. kódrészlet A C# nyelv while szerkezete

- *foreach* szerkezet: A foreach bevezetése rendkívül hasznos új eszköz volt a C# nyelv fejlődése során. Segítségével egy kollekció, például egy tömb vagy lista elemeit iterálhatjuk, azaz lépkedhetünk végig annak minden elemén. Ez a ciklus automatikusan kezeli az elemeket, és nem kell külön nyilvántartani az indexet.

```
foreach (int szam in szamok)
{
    Console.WriteLine(szam);
}
```

12. kódrészlet A C# nyelv foreach szerkezete

És most térjünk rá a C# és a .NET keretrendszer egyik leglényegesebb koncepcióira, az objektum-orientáltság, az osztályok és névtereik kérdésére, és ezzel már egy kicsit komolyabb területre tévedünk.

6.1.2.3. Objektum-orientált programozás

Az objektum-orientált programozás az egyik legelterjedtebb programozási paradigma, amely alapelvei a valós világ modellezését célozzák, az adatok és az ezekkel kapcsolatos műveletek szervezésére és kezelhetőségére fókuszálva. C# nyelvben az objektumorientált programozás kulcsfontosságú szerepet játszik, és az **osztályok** az alapvető építő kövei.

Az objektum-orientált programozás négy fő alapelven nyugszik: az absztrakció, az bezárás, az öröklődés és a polimorfizmus.

- **Absztrakció**: Az absztrakció lehetővé teszi, hogy a komplex rendszerek kezelhetőbbé váljanak azáltal, hogy a lényegtelen részleteket elrejtjük, és csak a fontos szempontokat emeljük ki. Például egy autó modellje esetén csak az autó működéséhez szükséges elemeket modellezzük, míg a belső mechanikai részletek el vannak rejtve.
- **Bezárás** (enkapszuláció): Az bezárás azt jelenti, hogy az adatokat és a hozzájuk tartozó műveleteket (metódusokat) egyetlen egységbe, azaz objektumba zárjuk. Ez segít megőrizni az adatok épségét, hiszen csak az objektumon belüli metódusokon keresztül érhetjük el őket.
- **Öröklődés**: Az öröklődés lehetővé teszi, hogy egy osztály más osztályoktól örökölje azok tulajdonságait és metódusait. Ez segít újra felhasználni a kódot, és csökkenti a redundanciát. Az alosztályok kiterjeszthetik és módosíthatják az ősosztályok viselkedését.
- **Polimorfizmus**: A polimorfizmus lehetővé teszi, hogy ugyanaz a művelet különböző típusú objektumok esetében különböző módon valósuljon meg. Ezáltal egy egységes interfészen keresztül többféle konkrét implementációt is használhatunk.

Osztály létrehozása

Az osztály definiálásához a *class* kulcsszót használjuk a C# nyelvben. Az alábbi példa egy egyszerű AUTÓ osztályt mutat be:

```
class Auto
{
    public string Marka { get; set; }
    public string Modell { get; set; }
    public int Evjarat { get; set; }

    public Auto()    {    }

    public Auto(string marka, string modell, int evjarat)
    {
        Marka = marka;
        Modell = modell;
        Evjarat = evjarat;
    }

    public void Indit()
    {
        Console.WriteLine($"{Marka} {Modell} indul.");
    }
}
```

13. kódrészlet Objektumosztály definiálása C# nyelv

Ebben a példában az *Auto* osztály három tulajdonságot tartalmaz: *Marka*, *Modell* és *Evjarat*, valamint egy két metódust, az osztály nevével egyező konstruktort és az *Indit* nevűt.

A konstruktorok az osztályok speciális metódusai, amelyek akkor hívódnak meg, amikor egy új objektumot hozunk létre. A konstruktor segítségével inicializálhatjuk az objektum adatait. A konstruktor neve mindig megegyezik az osztály nevével, de több is lehet belőle, különböző (akár nulla) paraméter számmal.

A *public* kulcsszóval ellátott tulajdonságok és metódusok kívülről is elérhetők az objektumokon keresztül, *private* kulcsszóval jelölt tulajdonság vagy metódus csak az osztály másik nevesített metódusából vagy a konstruktorból érhető el.

Objektum példányosítása

Az osztály definiálása után hozunk létre objektumokat. Az objektumok az osztály konkrét példányai, amelyek már valós adatokat és funkciókat tartalmaznak. Új objektumot a *new* kulcsszóval hozunk létre (példányosítunk):

```
Auto auto1 = new Auto();
auto1.Marka = "Toyota";
auto1.Modell = "Corolla";
auto1.Evjarat = 2020;
auto1.Indit();
Auto auto2 = new Auto("Ford", "Fiesta", 2018);
auto1.Modell = "Focus";
```

14. kódrészlet Objektumosztály példányosítása C# nyelven

Itt az *auto1* nevű objektum az *Auto* osztály (üres) példánya, amely utána kap konkrét értékeket, majd az *Indit* metódus segítségével a program kimenetére írja az autó indításának üzenetét. Az *auto2* nevű objektum létrehozásakor már beállításra kerülnek annak tulajdonságai a háromparaméterű konstruktor hívásával, majd a *Modell* tulajdonság felül lesz írva.

6.1.2.4. Névterek

A névterek (angolul: namespace) a C# nyelvben fontos szerepet játszanak a kód szervezésében és strukturálásában.

Amikor nagyobb méretű alkalmazásokat fejlesztünk, gyakran előfordulhat, hogy több különböző modulban ugyanazt a nevet szeretnénk használni egy osztályhoz vagy metódushoz. A névterek segítenek elkülöníteni ezeket, mivel minden elem egy adott névtérhez tartozik, és ezáltal egyedi módon azonosítható. A névtereket a *namespace* kulcsszóval hozzuk létre, és az elemeket ezen belül definiáljuk.

```
namespace Autoalkatreszek
{
    class Motor
    {
        public void Indit()
        {
            Console.WriteLine("A motor indul.");
        }
    }
}
```

15. kódrészlet Névtér definiálása C# nyelvben

Itt az *Autoalkatreszek* névtéren belül definiáltuk a *Motor* osztályt. Ha egy másik fájlban vagy névtérben szeretnénk ezt az osztályt használni, akkor vagy a teljesen kvalifikált nevet használjuk:

```
Autoalkatreszek.Motor motor = new Autoalkatreszek.Motor();
motor.Indit();
```

16. kódrészlet Névtérben definiált elemek felhasználása C# nyelvben (1)

Vagy importáljuk a névteret a using kulcsszóval:

```
using Autoalkatreszek;

Motor motor = new Motor();
motor.Indit();
```

17. kódrészlet Névtérben definiált elemek felhasználása C# nyelvben (2)

Az általános gyakorlat utóbbi megoldás alkalmazása, a plugin programkódjában is ezt fogjuk használni.

Ennyi C# alapozás után térjünk rá az AutoCAD programozási felületének ismertetésére.

6.2. ObjectARX.NET

Az AutoCAD-et az Autodesk fejlesztette ki, és először 1982-ben mutatták be. Az AutoCAD gyorsan népszerűvé vált a mérnökök, építészek és tervezők körében, mivel lehetővé tette a komplex rajzok gyors és pontos elkészítését számítógépen. Az AutoCAD első verziói viszonylag zárt rendszerek voltak, amelyek korlátozott lehetőségeket biztosítottak a felhasználói testre szabásra. Azonban az idő előrehaladtával az Autodesk látta, hogy a felhasználók részéről egyre nagyobb igény mutatkozik arra, hogy saját kiegészítőket és eszközöket fejlesszenek a szoftverhez.

Ez az igény vezetett az ObjectARX könyvtár kifejlesztéséhez. Az ObjectARX egy C++ alapú alkalmazásprogramozási interfész (API), amely lehetővé tette a fejlesztők számára, hogy mélyen integrálják saját alkalmazásaikat az AutoCAD-be. Az ObjectARX segítségével a fejlesztők hozzáférhetnek az AutoCAD belső adataihoz és parancsaihoz, valamint saját egyedi parancsokat, eszközöket és funkciókat hozhatnak létre.

A .NET keretrendszer megjelenése új lehetőségeket nyitott meg a fejlesztők számára. Az ObjectARX .NET verziója lehetővé tette, hogy a fejlesztők a C# és a Visual Basic .NET nyelveket használva dolgozzanak az AutoCAD-ben, ami sokkal egyszerűbbé tette a programozást és az alkalmazások fejlesztését. A .NET platform integrációja óriási

előnyt jelentett, mivel a fejlesztők így kihasználhatták a .NET által nyújtott előnyöket, mint például az egyszerűbb memória- és típuskezelést, valamint a gazdag könyvtárkészletet.

6.2.1. Az ObjectARX .NET szerepe és fontossága

Az ObjectARX .NET API a fejlesztők számára lehetőséget biztosít arra, hogy kibővítsék az AutoCAD funkcióit, és testre szabják azt a saját igényeik szerint. Az ObjectARX .NET interfész használata során a fejlesztők számos lehetőséggel találkoznak:

- **Parancsok testreszabása**: A fejlesztők képesek saját AutoCAD parancsokat létrehozni, amelyek pontosan az általuk szükséges funkciókat látják el. Ezek a parancsok lehetnek egyszerűek vagy bonyolultak, és lehetőség van arra, hogy integrálják őket az AutoCAD felhasználói felületébe.

- **Rajzobjektumok manipulálása**: Az AutoCAD-ben található rajzobjektumok (pl. vonalak, körök, ívek, poligonok) könnyen elérhetők és módosíthatók az ObjectARX .NET segítségével. Ez lehetőséget ad arra, hogy automatikusan generáljunk vagy módosítsunk rajzokat bizonyos kritériumok alapján.

- **Adatbáziskezelés**: Az AutoCAD minden egyes rajzot adatbázisként kezel, amely tartalmazza a különböző rajzobjektumokat és azok attribútumait. Az ObjectARX .NET lehetővé teszi, hogy a fejlesztők hozzáférjenek ehhez az adatbázishoz, és dinamikusan módosítsák azt.

- **Felhasználói felület kiterjesztése**: Az ObjectARX .NET API segítségével a fejlesztők egyedi eszköztárakat, menüket és ablakokat hozhatnak létre az AutoCAD felhasználói felületén, ezzel könnyítve a felhasználók munkáját.

Az ObjectARX .NET különösen fontos az olyan felhasználók és cégek számára, amelyeknek egyedi igényeik vannak, és amelyek nem találják meg az összes szükséges funkciót az AutoCAD alapvető eszköztárában. A testreszabott eszközök jelentős mértékben növelhetik a termelékenységet és hatékonyságot, mivel lehetővé teszik az automatizált feladatvégrehajtást és az ismétlődő feladatok egyszerűsítését.

6.2.2. Az ObjectARX .NET alapjai

Az ObjectARX .NET API használata előtt fontos megérteni annak alapvető komponenseit és szerkezetét. Az alábbiakban bemutatjuk az ObjectARX .NET néhány legfontosabb alapfogalmát és struktúráját:

1.) **AutoCAD fájlstruktúra**

Az AutoCAD-ben minden egyes rajz egy DWG formátumú fájlban van tárolva. Ez a fájl tartalmazza a rajz minden elemét, beleértve a vonalakat, köröket, blokkokat, attribútumokat és egyéb objektumokat. Az ObjectARX .NET segítségével a fejlesztők közvetlenül hozzáférhetnek ezekhez az adatokhoz, és módosíthatják azokat.

2.) **Adatbázis**

Az AutoCAD minden egyes rajzot egy fa struktúrájú adatbázisnak tekint, ez az adatbázis tartalmazza a rajz összes objektumát és azok tulajdonságait, de ez az adatszerkezet nem tévesztendő össze a valódi adatbázisokkal.
Az ObjectARX .NET lehetőséget biztosít arra, hogy közvetlenül manipuláljuk ezeket az adatokat, például lehetőség van arra, hogy új objektumokat hozzunk létre, módosítsuk a meglévő objektumokat, vagy töröljük azokat.

3.) **Tranzakciók**

Az ObjectARX .NET programozási modellje tranzakciókat használ az adatbázis-módosítások kezelésére. Egy tranzakció egy logikai műveletsorozatot jelöl, amelyben az összes műveletet vagy sikeresen végrehajtják, vagy ha hiba történik, akkor az összes módosítást visszavonják. Ez a megközelítés biztosítja, hogy az adatbázis mindig konzisztens állapotban maradjon, még akkor is, ha valamilyen probléma lép fel a módosítás során.

4.) **Parancsok**

Az ObjectARX .NET lehetőséget biztosít arra, hogy a fejlesztők saját parancsokat hozzanak létre. Ezek a parancsok ugyanúgy működnek, mint az AutoCAD beépített parancsai, és integrálhatók az AutoCAD parancssorába. A fejlesztők saját algoritmusokat és logikát adhatnak ezekhez a parancsokhoz, így testreszabott funkciókat hozhatnak létre.

5.) **Eseménykezelés**

Az ObjectARX .NET API támogatja az eseménykezelést, ami azt jelenti, hogy a fejlesztők reagálhatnak különböző eseményekre, amelyek az AutoCAD-ben történnek. Például egy fejlesztő kódot írhat arra, hogy automatikusan végrehajtson egy műveletet, amikor a felhasználó létrehoz vagy módosít egy objektumot.

Az alábbiakban egy egyszerűbb példát mutatunk be, mely a felhasználó bekért adatok alapján hoz létre egy új kör objektumot:

```
[CommandMethod("DrawCircle")]
 public void DrawCircle()
 {
    Document acDoc = Application.DocumentManager.MdiActiveDocument;
    Database acCurDb = acDoc.Database;
    Editor ed = acDoc.Editor;
    PromptPointResult pPtRes = ed.GetPoint("\nVálasz középpontot:");
    if (pPtRes.Status != PromptStatus.OK) return;
    PromptDoubleResult pRadRes = ed.GetDistance("\nAdd meg a kör sugarát:",
pPtRes.Value);
     if (pRadRes.Status != PromptStatus.OK) return;
     using (Transaction tr = acCurDb.TransactionManager.StartTransaction())
     {
        BlockTable acBlkTbl = tr.GetObject(acCurDb.BlockTableId,
OpenMode.ForRead) as BlockTable;
        BlockTableRecord acBlkTblRec =
tr.GetObject(acBlkTbl[BlockTableRecord.ModelSpace], OpenMode.ForWrite) as
BlockTableRecord;
        Circle acCircle = new Circle(pPtRes.Value, Vector3d.ZAxis,
pRadRes.Value);
         acBlkTblRec.AppendEntity(acCircle);
         tr.AddNewlyCreatedDBObject(acCircle, true);
         tr.Commit();
     }
     ed.WriteMessage("\nKör megrajzolva a megadott középponttal és
sugárral.");
 }
```

18. kódrészlet ObjectARX .NET példa

A *DrawCircle* függvényt a *CommandMethod* attribútumban megadott névvel lehet parancsként meghívni az AutoCAD programban.

A program elején az AutoCAD dokumentum, annak adatbázisa és parancsorának példánya kerül eltárolásra, majd a két prompt (bevitel) függvény hívással egy pontot (*ed.GetPoint*) és egy attól való távolságot (*ed.GetDistance*) kérünk be a felhasználótól.

Ezután a rajzi adatbázis kerül megnyitásra egy új tranzakcióra, és az adatbázis rajzi objektumokat tároló ún. *BlockTable* részéhez egy új *Circle* (kör) objektumot adunk, majd végül a *tr.Commit* függvény hívásával jóváhagyjuk a tranzakciót, így a kör véglegesen bekerül a rajzba.

7. Az AutoCAD Map3D funkcióbővítése a C# programozási nyelv és ObjectARX.NET segítségével

A fenti rövid példa hagyományos rajzi elemek hozzáadását, kezelését mutatta be.

Sajnos az ESZTER programban a távközlési hálózati elemek egy másik fajta adatbázisban, az AutoCAD Map 3D szakági adatmodell SQLite adatbázisában kerülnek tárolásra, így más eszközöket kell használni.

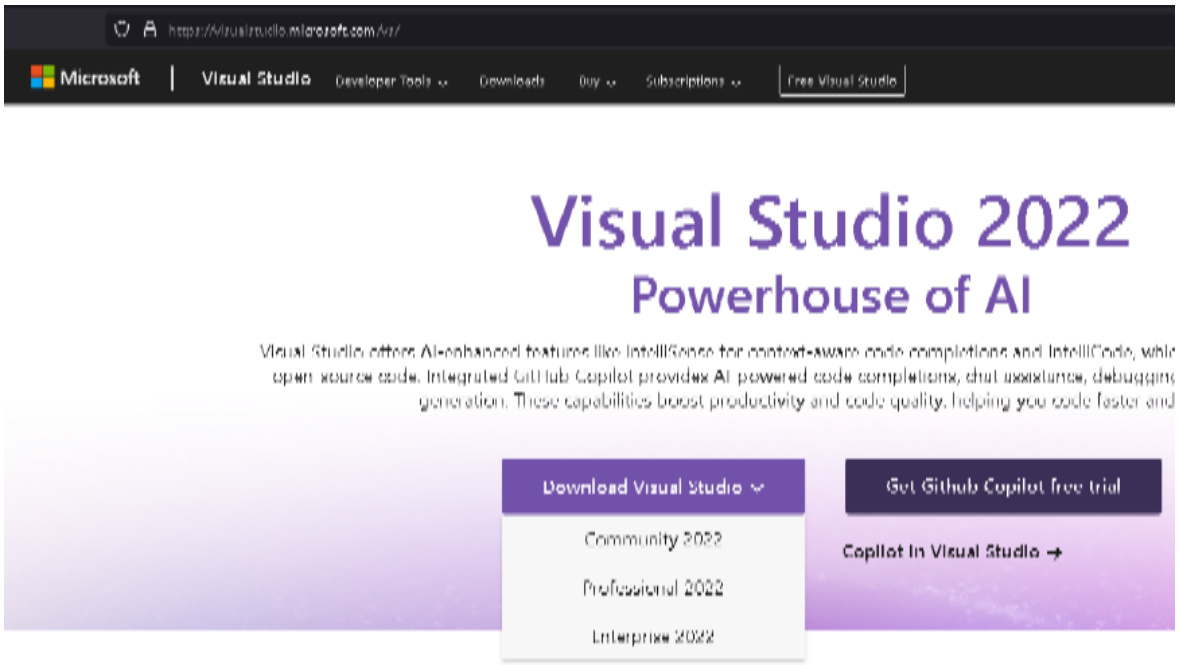
A szakági adatmodell tulajdonképpen a MapGuide térinformatikai adatbázis beágyazása az AutoCAD DWG állományába, így egyértelmű, hogy annak programozási eszköztárát is szükséges majd felhasználnunk.

7.1. A szükséges programok illetve fejlesztőeszközök telepítése és beállítása

A fejezetben bemutatandó példákat a Microsoft Visual Studio integrált fejlesztői környezetét (integrated development environment, IDE) fogjuk használni, valamint az AutoCAD ObjectARX alkalmazásprogramozási interfészt, annak is a .NET keretrendszerhez illesztett változatát, továbbá az ehhez kiadott Visual Studio projektvarázslót.

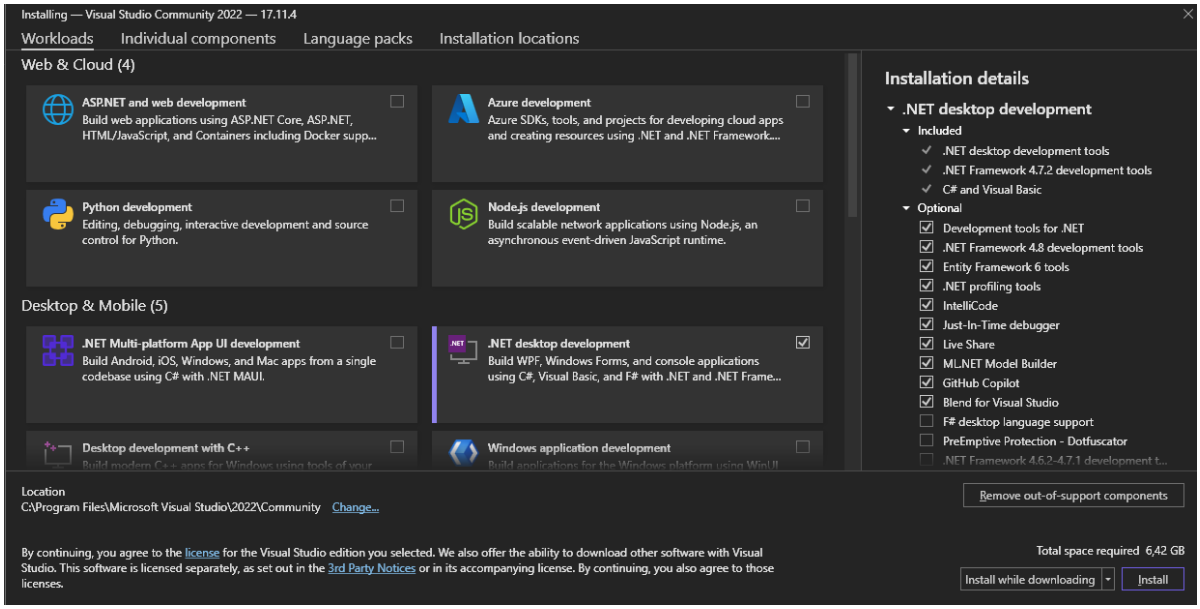
7.1.1. A Visual Studio telepítése

A Visual Studio-t https://visualstudio.microsoft.com lehet letölteni. Tekintettel, hogy a program alapvetően fejlesztőcégek részére készült, így a elég borsos árral rendelkeznek a professzionális, és a programozók együttműködését segítő változatok. Szerencsére rendelkezésre áll egy ingyenes változat is, mely egyedi fejlesztők és tanulók részére teljesen megfelelő, ez a Visual Studio Community.



2. ábra A Visual Studio Community letöltése (forrás: Microsoft)

Letöltés után a telepítőt elindítva, számunkra csak a .NET desktop development eszközcsoport telepítése válik szükségessé.



3. ábra A Visual Studio Community telepítése, a .NET desktop eszközökkel

7.1.2. Az AutoCAD ObjectARX és a DotNet Wizard telepítése

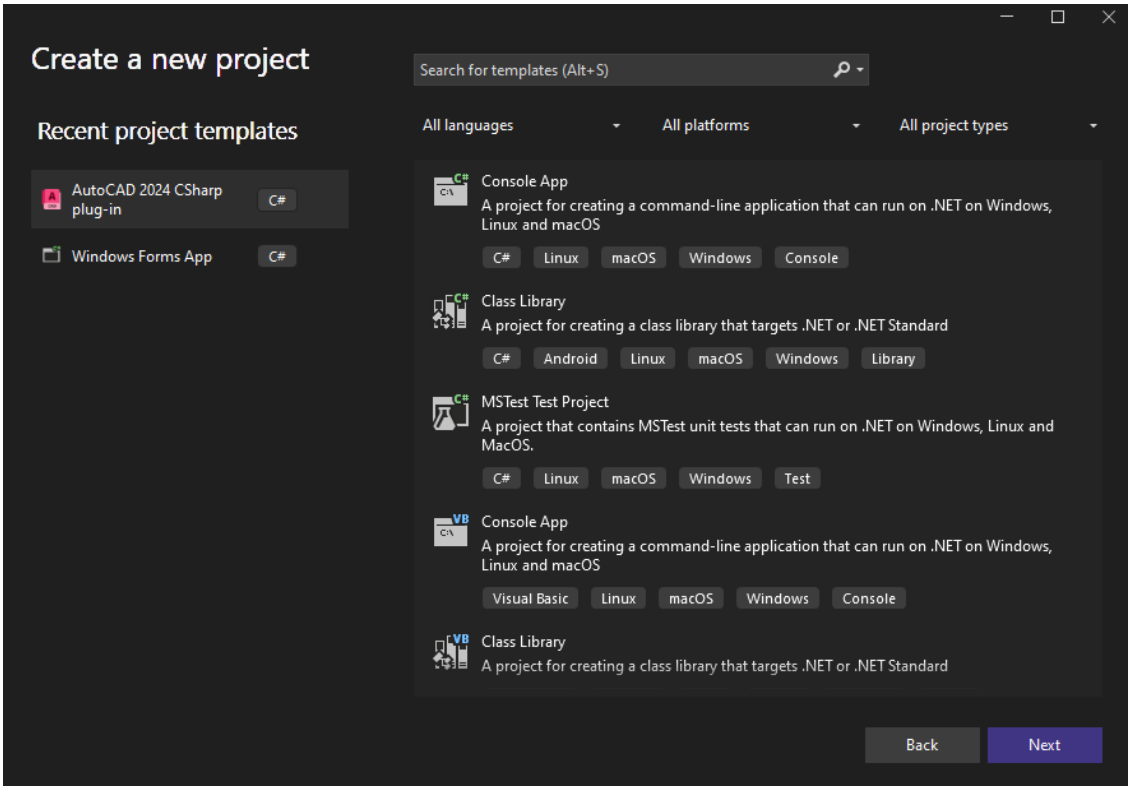
Ezután le kell tölteni és telepíteni kell a megfelelő verziójú ObjectARX SDK-t a https://aps.autodesk.com/developer/overview/autocad-objectarx-sdk-downloads oldalról. Az ESZTER jelenlegi verziója a 2024-es AutoCAD-re épül, így ennek megfelelő verziót kell telepíteni.

(Megjegyzem, az AutoCAD API a 2021-es verzió és a 2024-es verzió között nem változott, azaz bármely verzióhoz készült plugin kompatibilis ezen összes verzióval.)

Az SDK telepítése után az AutoCAD DotNet Wizards-ot kell letölteni, a https://aps.autodesk.com/developer/overview/autocad oldal *Download past AutoCAD DotNet Wizards* részéről.

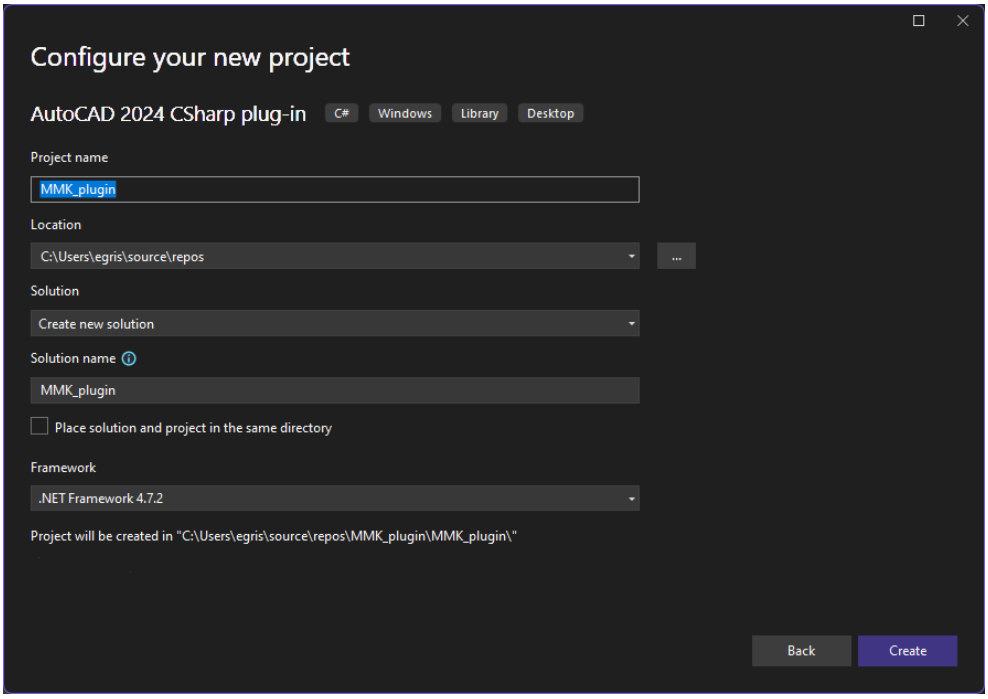
7.1.3. Új AutoCAD C# plugin projekt létrehozása

A varázsló telepítése után a Visual Studio-t elindítva, majd a Create New Project gombra kattintva, kiválasztható lesz az AutoCAD 2024 C# plug-in.



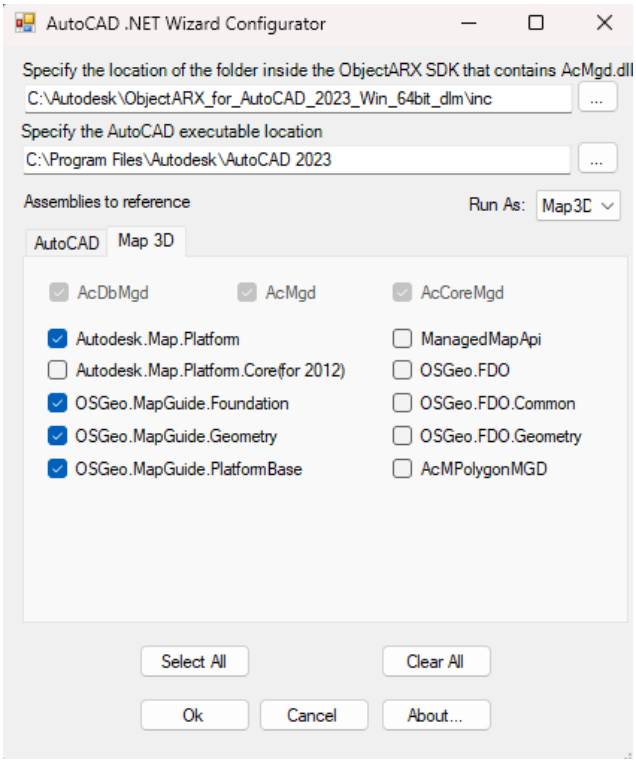
4. ábra A Visual Studio Community új projekt létrehozása

A *Next* gombot megnyomva a projekt beállításainak párbeszédablakába kerülünk, itt adhatjuk a projekt nevét és egyéb paramétereket. Legyen a projekt neve **MMK_plugin**, minden más maradjon alapértelmezett.



5. ábra Az AutoCAD DotNet Wizards beállításai és használata

A *Create* gombot megnyomva a varázsló beállításaihoz kerülünk.

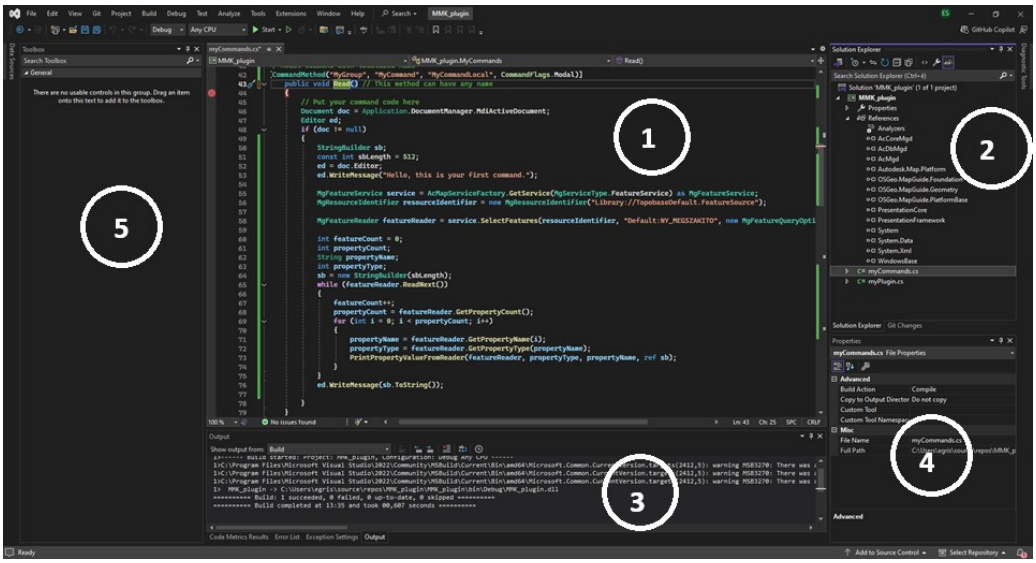


6. ábra Az AutoCAD DotNet Wizards beállításai és használata

A *Run As* legördölő menüben a Map 3D-t választva a varázsló a számunkra szükséges MapGuide függvénykönyvtárakat is beemeli az új projektbe.

7.1.4. A Visual Studio felülete és opcionális beállítások

Az *OK* gombra kattintva, a varázsló létrehozza a projektet, majd megnyílik a Visual Studio program felülete.

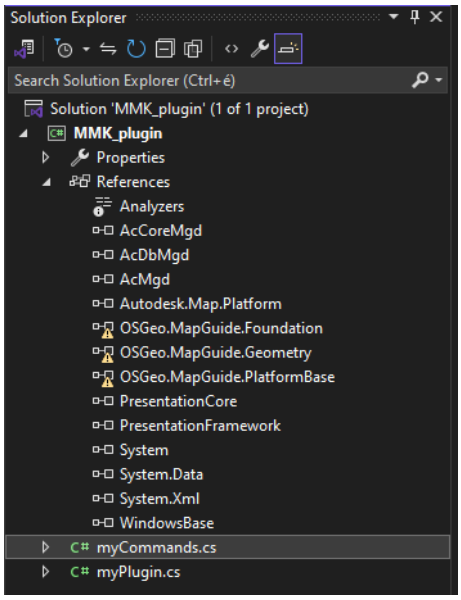


7. ábra Az Visual Studio felülete

Az **1. ablak** maga a kódszerkesztő, ahol az aktuális programkód és egyéb fájl szerkeszthető, a fájl tartalmának (programnyelv, stb.) megfelelő szemléletes szintaktikai színkódolással.

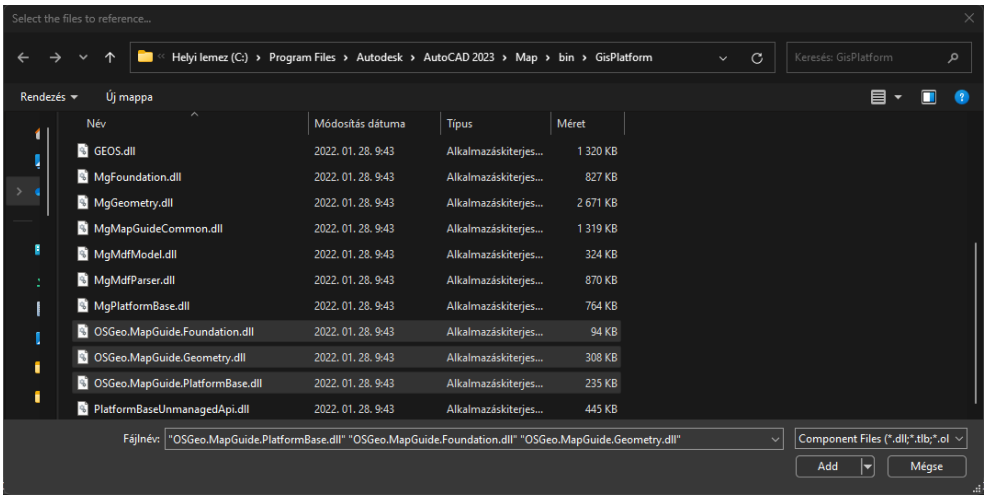
A **2. ablak** a Solution Explorer, a projekt összes állományát illetve referenciáit tartalmazza. Az egyes programkód állományokra kattintva azok az 1. ablakban nyílnak meg.

Amennyiben az új projekt létrehozásakor a referenciák sárga felkiáltójeles háromszöggel vannak jelölve, úgy azokat a rendszer nem találja az alap elérési útvonalakon.



8. ábra Visual Studio Solution Explorer ablak, hiányzó referenciákkal

Ekkor a *References*, jobb klikk, *Add References* menüpontot választva lehet azokat hozzáadni. A MapGuide hiányzó DLL moduljai az alábbi helyen érhetőek el:



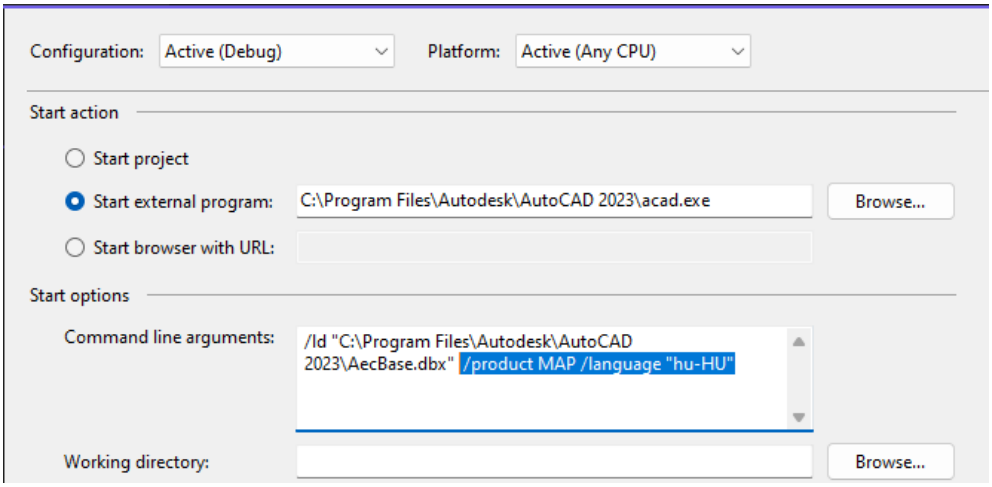
9. ábra Hiányzó MapGuide referencia DLL modulok elérési útvonala

A **3. ablak** az Output, a projekt programnyelvi és egyéb hibáinak üzenetei itt jelennek meg.

A **4. ablak** az Properties, tulajdonságok ablak, míg a **5. ablak** a Toolbar, eszköztár. Az általunk elkészítendő AutoCAD plugin elkészítése során nem használjuk, grafikus felületű program (beleértve AutoCAD palettákat, mint amilyen az ESZTER paneljei) készítésekor van jelentőségük.

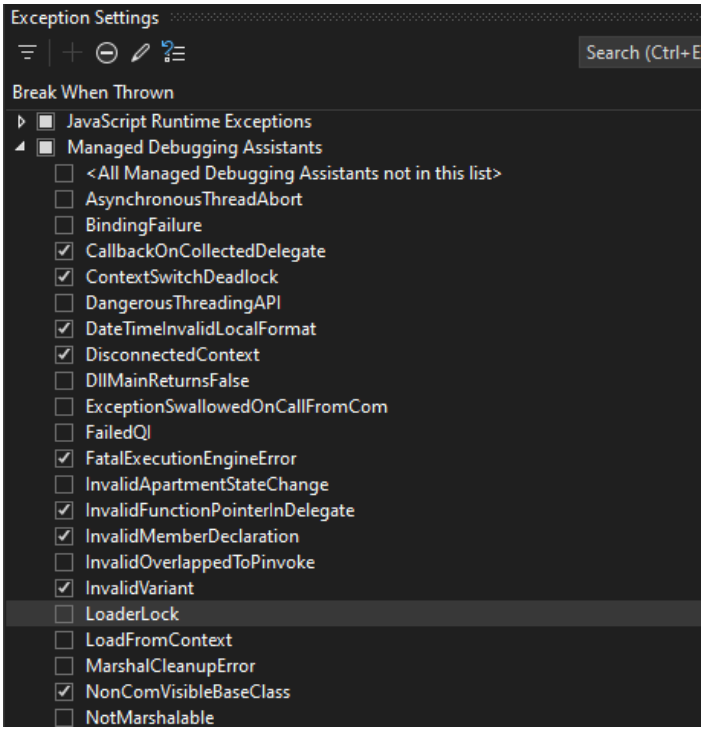
Még két dolgot szükséges módosítanunk a projekt és a Visual Studio beállításaiban.

Elsőként ellenőrizzük a projekt beállításaiban, hogy a projekt indításakor az AutoCAD-et Map 3D-ként indul-e el. A *Debug* menü *MMK_plugin Debug Properties* pontját választva az alábbi konfigurációs ablak töltödik be.



10. ábra Projekt Debug beállítások

Végül ahhoz, hogy a lefordított programmodulja teszteléséhez a Visual Studio Debug eszköztárával tudjuk az AutoCAD Map 3D-t elindítani, ki kell kapcsolni az ún. *LoaderLock* kivétel figyelését, melyet a *Debug* menü *Windows* almenü *Exception Setting* pontját választva, a megnyíló beállítási ablakban a *Managed Debugging Assistants* pont alatt kell kikapcsolni.



11. ábra LoaderLock kivétel figyelés kikapcsolása

7.2. Szakági adatmodell jellemzőosztályok (adatbázis táblák) kezelése C# nyelven

Az új projekt létrehozása és a szükséges beállítások elvégzése után rátérhetünk a segédlet legfontosabb részére, a Map 3D (és így az ESZTER) szakági adatmodelljének lekérdezési és módosítási lehetőségeire programozási eszközök felhasználásával.

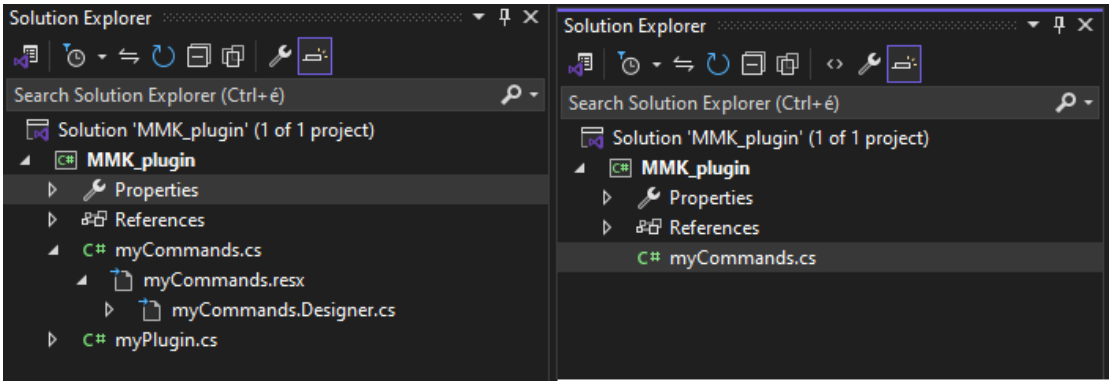
Mivel a szakági adatmodell tulajdonképpen egy SQLite adatbázis beágyazva a DWG állományba, az relációs adatbázisok négy fő műveletét kell megvalósítanunk az elkészítendő beépülő modulban:

- *SELECT*: az adatbázis tábla lekérdezése, vagy a tábla összes sorának vagy egy megadott szűrőfeltétel által kiválasztott sorok visszaadásával.
- *INSERT*: Új sor hozzáadása az adattáblához
- *UPDATE*: Megadott feltételnek megfelelő értéket tartalmazó sorok módosítása
- *DELETE*: Az adattábla azon sorainak törlése, mely a megadott feltételnek megfelelnek.

Így tehát négy parancs/függvény elkészítése szükséges, melyek a MapGuide alkalmazásprogramozási felületét fogják használni.

7.2.1. Az MMK plugin felépítése

Az elkészítendő plugin előkészítéséhez a Solution Explorerben le kell nyitni a *myCommands.cs* állomány listáját, és törölni kell a *myCommands.Designer.cs*, a *myCommand.resx*, és a teljes *myPlugin.cs* állományt.



12. ábra Projekt struktúra átalakítása

Ezután a myCommands.cs állományban töröljünk minden megjegyzést (a zöld színnel megjelenített „//” jelek utáni részek), illetve egyéb kódrészeket, hogy csak ennyi maradjon:

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.Runtime;
using System;

[assembly: CommandClass(typeof(MMK_plugin.MyCommands))]

namespace MMK_plugin
{
    public class MyCommands
    {
        [CommandMethod("MyCommand")]
        public void MyCommand()
        {
            Document doc =
Application.DocumentManager.MdiActiveDocument;
            Editor ed;
            if (doc != null)
            {
                ed = doc.Editor;
                ed.WriteMessage("Hello, this is your first command.");
            }
        }

    }
}
```

19. kódrészlet Az MMK plugin kiinduló kódstruktúrája

Nevezzük át a *MyCommands* osztályt *MapFeatureCommands* –ra, az osztályneven jobb klikk és **Rename** menüpontot választva. A Visual Studio beépített Rename funkciója nem csak az adott helyen írja át a nevet, hanem az összes kapcsolódó helyen, többek között a programkód állomány nevében is.

Ezután a using direktívák után adjuk hozzá a két, Map 3D és szakági modell, illetve a sztringek kezeléshez szükséges névteret:

```
using OSGeo.MapGuide;
using Autodesk.Gis.Map.Platform;
using System.Text;
```

20. kódrészlet A szükséges névterek hozzáadása

Az osztály elejére, a CommandMethod attribútum elé írjuk be az alábbi közös felhasználású MapGuide objektumok inicializálását elvégző kódot:

```
    private MgAgfReaderWriter agfReaderWriter =
                                            new MgAgfReaderWriter();
    private MgWktReaderWriter wktReaderWriter =
                                            new MgWktReaderWriter();
    private MgFeatureService service =
AcMapServiceFactory.GetService(MgServiceType.FeatureService) as
MgFeatureService;
    private MgResourceIdentifier resourceIdentifier =
new MgResourceIdentifier("Library://TopobaseDefault.FeatureSource");
```

21. kódrészlet MapGuide objektumok inicializálása

Első kettő a geometriai adatok kezelésére szolgáló objektumok, a harmadik a jellemző osztályok (feature) kezelését végző AutoCAD Map 3D belső adatbázis szolgáltatáshoz ad hozzáférést, a negyedik pedig magának az SQLite adatbázisnak az belső azonosítójára hivatkozást.

(Egy ESZTER rajzi állományban a *Tervezés és elemzés* munkaterületen, a Térkép beállításai szalagon a *Kapcsolódás* menüben láthatjuk az alapértelmezett SQLite kapcsolatot, melynek neve **TopobaseDefault**)

7.2.2. Szakági adatmodell jellemzőosztály adatok listázása (ReadFeatures)

A jellemzőosztályok listázására egy parancssoros alkalmazás készült, azaz az AutoCAD parancssorában kérdezi meg a lekérdezendő jellemzőosztály nevét, illetve ugyanide listázza ki az szöveges formában.

A teljes MyCommand függvény kódját cseréljük le az alábbira:

```
[CommandMethod("ReadFeautures")]
    public void ReadFeatures()
    {

        Document doc = Application.DocumentManager.MdiActiveDocument;
        Editor ed;
        if (doc != null)
        {
            StringBuilder sb;
            const int sbLength = 512;
            ed = doc.Editor;
            PromptResult fName =
                        ed.GetString("Válassz jellemzőosztály nevet");
            if (fName.StringResult.Length == 0 ) return;

            MgFeatureReader featureReader =
service.SelectFeatures(resourceIdentifier,
fName.StringResult.ToUpper(), new MgFeatureQueryOptions());
```

22. kódrészlet ReadFeatures függvény (1. rész)

```
            int featureCount = 0;
            int propertyCount;
            String propertyName;
            int propertyType;
            sb = new StringBuilder(sbLength);

             while (featureReader.ReadNext())
            {
                featureCount++;
                propertyCount = featureReader.GetPropertyCount();
                for (int i = 0; i < propertyCount; i++)
                {
                    propertyName = featureReader.GetPropertyName(i);
                    propertyType =
                        featureReader.GetPropertyType(propertyName);
                    PrintPropertyValueFromReader(featureReader,
propertyType, propertyName, ref sb);
                }
                sb.AppendLine();
            }
            ed.WriteMessage(sb.ToString());
        }
    }
```

23. kódrészlet ReadFeatures függvény (2. rész)

Az új parancs függvényünk megvizsgálja, van-e aktív dokumentum megnyitva (azaz egy megnyitott rajzból hívjuk-e), majd a paracssorban egy sztring prompt (*ed.GetString*) segítségével rákérdez a listázandó jellemzőosztály nevére.

Nem üres válasz esetén (ha az *fName* promptválasz változó *StringResult* eredmény érték hossza nem nulla) folytatódik a program egy új MapGuide *MgFeatureReader* objektum létrehozásával.

A MgFeatureReader objektumot az adatbázis szolgáltatás objektum SelectFeatures metódusa adja vissza, azt 3 paraméterrel meghívva. Az első a már említett SQLite adatbázis azonosító, a második maga a lekérdezendő jellemzőosztály neve, míg a harmadik a lekérdezés szűrésére szolgáló paraméter objektum. Esetünkben ez üres, azaz a konstruktorát paraméterek nélkül hívjuk, mivel a teljes adattáblát ki akarjuk listázni.

A jellemzőosztály nevei az adatbázisban mindig csupa nagybetűből állnak, így elkerülendő felhasználói bevitel kisbetű-nagybetű tévesztését, a megadott jellemzőnevet (*fName.StringResult* értéke) a sztring adattípusok beépített *ToUpper* függvényével alakítjuk csupa nagybetűssé.

Ezután pár egyéb változó definiálása után egy új *StringBuilder* objektumot hozunk létre. A *StringBuilder* osztály a sztringek összefűzésének, módosításának memóriahatékony eszköze. (a C# alap string típusa létrehozása után nem módosítható, annak értékének bármely módosítása új objektum példányosítását jelenti.)

Egy *while* ciklusban a kiválasztott jellemzőosztály minden adatsorát kiolvassuk, amíg a *featureReader. ReadNext* metódusa igaz értéket ad, azaz volt még adatsor.

A *while* cikluson belül lekérdezzük az adott adatsor mezőinek (oszlopainak) számát (*featureReader.GetPropertyCount*), majd ezt felhasználva egy nulla kezdőértékkel inicializált *for* ciklusban az i munkaváltozó értékének megfelelő számú mezőnevet kérdezzük le (*GetPropertyName*), ezt felhasználva pedig a mező típusát (*GetPropertyType*), végül ezen értékek segítségével hívjuk a *PrintPropertyValueFromReader* nevű segédfüggvényünket, melynek kódrészlete:

```
private void PrintPropertyValueFromReader(MgReader reader, int
propertyType, String propertyName, ref StringBuilder sb)
{
    Boolean isNull = reader.IsNull(propertyName);
    if (propertyType == MgPropertyType.Boolean)
    {
        sb.Append(propertyName + " = Boolean(" + (isNull ? "null" :
reader.GetBoolean(propertyName).ToString()) + ")\t");
    }
    else if (propertyType == MgPropertyType.Double)
    {
        sb.Append(propertyName + " = Double(" + (isNull ? "null" :
reader.GetDouble(propertyName).ToString()) + ")\t");
    }
    else if (propertyType == MgPropertyType.Geometry)
    {
        sb.Append(propertyName + " = Geometry(" + (isNull ? "null" :
MgByteReaderToWktText(reader.GetGeometry(propertyName))) + ")\t");
    }
    else if (propertyType == MgPropertyType.Int32)
    {
        sb.Append(propertyName + " = Int32(" + (isNull ? "null" :
reader.GetInt32(propertyName).ToString()) + ")\t");
    }
    else if (propertyType == MgPropertyType.Int64)
    {
        sb.Append(propertyName + " = Int64(" + (isNull ? "null" :
reader.GetInt64(propertyName).ToString()) + ")\t");
    }
    else if (propertyType == MgPropertyType.String)
    {
        sb.Append(propertyName + " = String(" + (isNull ? "null" :
reader.GetString(propertyName)) + ")\t");
    }
    else
    {
        sb.Append(propertyName + " = Egyéb(" + propertyType.ToString()
+ ")\t");
    }
}
```

24. kódrészlet PrintPropertyValueFromReader függvény

A függvény megkapja paraméterként a *featureReader* objektumok, a nevét, típusát és ún. referencia paraméterként a *StringBuilder* objektumot.

A függvényben az adatmezők típusát vizsgáló **if-elseif-if** feltételszerkezetben az egyes típusok esetén a *featureReader* objektum különféle metódusaival olvassuk ki a mezők konkrét értékeit, majd fűzzük hozzá a StringBuilder.

A geometriai adatok kiolvasásához (*MgPropertyType.Geometry*) egy másik segédfüggvényt használunk.

```
private String MgByteReaderToWktText(MgByteReader byteReader)
{
    MgGeometry geometry = agfReaderWriter.Read(byteReader);
    String wktText = wktReaderWriter.Write(geometry);
    return wktText;
}
```

25. kódrészlet MgByteReaderToWktText függvény

Az adatbázisban a geometriai adatok bináris formában vannak tárolva (a *featureReader* ez esetben ezt adja vissza), a segédfüggvényben ezt kiolvassuk az *agfReaderWriter.Read* metódus segítségével, majd a *wktReaderWriter.Write* metódussal String típusuvá alakítjuk, és ezt adjuk vissza a függvény visszatérési értékének.

Ha kész vagyunk a kód szerkesztésével, az elkészült parancsfüggvényt le tudjuk tesztelni. Ehhez a Visual Studio eszköztárán a 

Start gombot megnyomva elindul az AutoCAD Map 3D program, melyben egy korábbi ESZTER rajzot megnyitva, a NETBETÖLT paranccsal kell a lefordított plugint betölteni.

Alapértelmezett esetben ez a C:\Users*felhasználónév*\source\repos könyvtárban, a MMK_plugin\MMK_plugin\bin\Debug alkönyvtárban a lesz megtalálható.

Betöltése után kiadva a *ReadFeatures* parancsot, és például a „NY_MEGSZAKITO” nevet megadva jellemzőosztálynak, az eredmény valami hasonló:

13. ábra ReadFeature parancs eredménye

7.2.3. Jellemzőosztály adatok módosítása (UpdateFeatures)

Következő feladat a meglévő a jellemzőosztályok értékeinek módosítása.

Ehhez már alaposan ismerni kell az ESZTER (vagy más AutoCAD Map 3D szakági modellként használt adatbázis) felépítését.

Egyszerű példaként a megszakító objektumok (NY_MEGSZAKITO tábla) beépítési év mezőjét módosítjuk 2024-ről 2025-re.

```
[CommandMethod("UpdateFeautures")]
public void UpdateFeatures()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Editor ed;
    if (doc != null)
    {
        ed = doc.Editor;
        ed.WriteMessage("Megszakítók beépítési évének módosítása 2024-
ről 2025-re");
        MgFeatureCommandCollection mgFeatureCommandCollection = new
MgFeatureCommandCollection();
        MgPropertyCollection mgPropertyCollection = new
MgPropertyCollection();

        mgPropertyCollection.Add(new MgInt32Property("BEEPEV", 2025));
        mgFeatureCommandCollection.Add(new
MgUpdateFeatures("NY_MEGSZAKITO", mgPropertyCollection, "BEEPEV =
2024"));
        service.UpdateFeatures(resourceIdentifier,
mgFeatureCommandCollection, true);

    }
}
```

26. kódrészlet UpdateFeatures függvény

Az előző példához hasonlóan, csak megnyitott DWG esetén fut le a parancs, kiírja tájékoztatásként a felhasználónak, hogy mi is fog történni.

Ezután egy-egy *MgFeatureCommandCollection* és *MgPropertyCollection* kerül létrehozásra. Előbbi több (ugyanolyan) módosítási parancs összegyűjtésére szolgál, tehát esetünkben az NY_MEGSZAKITO tábla mellett még más adattáblákat is lehetne egy paranccsal módosítani.

Utóbbi pedig a módosítandó mezők gyűjteménye, esetünkben csak egy mező módosítása történik, az Int32 típusú BEEPEV mezőé, új értéke 2025 lesz (MgInt32Property).

Következő lépésben új *MgUpdateFeatures* kerül hozzáadásra a parancskollekcióhoz. Az *NY_MEGSZAKITO* táblán, az *mgPropertyCollection* mezőkollekcióban összegyűjtött módosításokkal és az utolsó, *String* típusú szűrőfeltétellel kerül létrehozásra

Végül az adatbázis szolgáltatás *service.UpdateFeatures* metódusával kerül véglegesítésre a módosítás.

A fenti példa csak rövid bemutatása a módosítási (Update) eszközöknek. Természetesen ettől jóval összetettebb szűrőfeltételeket is lehet alkalmazni, az SQL nyelv összes operátorával (=, <, >, LIKE, stb.), illetve a módosítandó mezők típusának megfelelő Property objektumokkal (pl. *MgInt64Property*, *MgStringProperty*, stb.)

.

7.2.4. Jellemzőosztály adatok törlése (DeleteFeatures)

Következő lépés a már meglévő adattábla sorok (jellemzők) törlése.

Itt szintén szükséges az adatbázis alaposabb ismerete.

Példaként az előző alfejezetben módosított megszakító objektumok (NY_MEGSZAKITO tábla) kerülnek törlésre (beépítési évük 2025).

```
[CommandMethod("DeleteFeautures")]
public void DeleteFeatures()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Editor ed;
    if (doc != null)
    {
        ed = doc.Editor;
        ed.WriteMessage("2025 beépítési évű megszakítók törlése");
        MgFeatureCommandCollection mgFeatureCommandCollection = new
MgFeatureCommandCollection();
        mgFeatureCommandCollection.Add(new
MgDeleteFeatures("NY_MEGSZAKITO", "BEEPEV=2024"));
        service.UpdateFeatures(resourceIdentifier,
mgFeatureCommandCollection, true);
    }
}
```

27. kódrészlet DeleteFeatures függvény

A kód ránézésre hasonló, bár egy kicsit egyszerűbb mint az Update függvényben alkalmazott.

Az eleje ugyanaz, ugyanazokkal a feltételvizsgálatokkal és üzenetkiírással, parancskollekció definiálással.

Itt nincs szükség Property mező objektumokra és kollekciójukra, csak egy új *MgDeleteFeatures* objektumra, mely az adattábla nevét és a szűrőfeltételt kapja paraméterül.

Az utolsó sor ugyanaz, tekintettel, hogy az törlés is tulajdonképpen módosítás (update) az adatbázis szempontjából.

7.2.5. Jellemzőosztályokhoz új adat hozzáadása (AddFeatures)

Végül, de nem utolsó sorban új szakági modell objektum hozzáadása szükséges, hogy teljes legyen az alapműveletek lefedése.

Természetesen a koncepció hasonló az Update függvényben alkalmazottakhoz.

```
[CommandMethod("AddFeautures")]
public void AddFeatures()
{
    Document doc = Application.DocumentManager.MdiActiveDocument;
    Editor ed;
    if (doc != null)
    {
        ed = doc.Editor;
        PromptPointResult pPtRes = ed.GetPoint("\nVálasz
középpontot:");
        if (pPtRes.Status != PromptStatus.OK) return;
        MgFeatureCommandCollection mgFeatureCommandCollection = new
MgFeatureCommandCollection();
        MgPropertyCollection mgPropertyColl = new
MgPropertyCollection();
        MgGeometryFactory geomFactory = new MgGeometryFactory();
        MgCoordinate coord =
geomFactory.CreateCoordinateXYZ(pPtRes.Value.X, pPtRes.Value.Y, 0);
        MgPoint mgPoint = geomFactory.CreatePoint(coord);
        MgByteReader mgByteReader = agfReaderWriter.Write(mgPoint);
        mgPropertyColl.Add(new MgGeometryProperty("GEOM",
mgByteReader));
        mgPropertyColl.Add(new MgDoubleProperty("ORIENTATION", 0));
        mgPropertyColl.Add(new MgInt32Property("OSZ_TIPUS", 504));
        mgPropertyColl.Add(new MgInt32Property("OSZ_ANYAG", 4));
        mgPropertyColl.Add(new MgInt32Property("OSZ_MAGASAG", 10));
        mgPropertyColl.Add(new MgInt32Property("EHO_OBJ", 608));
        mgPropertyColl.Add(new MgInt32Property("TERV_ID",
1111111111));
        mgPropertyColl.Add(new MgInt32Property("BEEPEV", 2024));
        mgPropertyColl.Add(new MgInt32Property("OSZ_ELEKTROMOS", 0));
        mgFeatureCommandCollection.Add(new
MgInsertFeatures("NY_OSZLOP", mgPropertyColl));
        service.UpdateFeatures(resourceIdentifier,
mgFeatureCommandCollection, true);
    }
}
```

28. kódrészlet AddFeatures függvény

A függvény elején a GetPoint prompt segítségével bekérünk egy beillesztési pontot, majd szokásos *MgFeatureCommandCollection* és *MgPropertyCollection* objektumok után egy *MgGeometryFactory*-t példányosítunk.

Ennek *CreateCoordinateXYZ* és *CreatePoint* metódusai segítségével hozunk létre egy új *MgPoint* objektumot a felhasználó által megadott pont válasz koordinátái alapján, természetesen az EHO-ban elvárt 0 Z koordinátával.

A *agfReaderWriter.Write* metódus segítségével a pont objektumot bináris formátumba konvertáljuk, majd hozzáadásra kerül a GEOM mezőhöz, mint geometriai tulajdonság (*MgGeometryProperty*).

A rákövetkező sorok a különféle leíró mezők megfelelő típusú tulajdonságként kerülnek a kollekcióhoz hozzáadásra, végül a szokásos módon szintaktikával, a MgInsertFeatures metódussal kerül az új adat hozzáadásra a táblához.

A fenti példában a fix értékekkel kerülnek a leíró adatok hozzáadásra az új adatsorhoz, de persze ez lehetne interaktívan, vagy grafikus felülettel megoldani, természetesen ez jóval hosszabb programkódot eredményezne és túlmutat jelen segédlet keretein.

8. Zárszó

Jelen segédlet viszonylag összetett és viszonylag hosszú, ennek ellenére a felmerült témakör nagyon kicsi szeletét tudta csak lefedni.

A példakódokat természetesen ki lehetne egészíteni kivételkezeléssel, de ennek koncepcióját kellene előbb tárgyalni. Ugyancsak át lehetne alakítani a kódokat, hogy sokkal rugalmasabbak legyenek, fix mezőnevek helyett az adattábla sémáját lekérdezve lehetne univerzálisabb függvényeket írni.

Nem esett továbbá szó arról, hogy lehet grafikus felületet létrehozni a parancsok futtatására és adatok bevitelére illetve megjelenítésére, legyen akár különálló Winforms illetve WPF technológiával készült ablakban, AutoCAD paletták felületén.

Ugyancsak említésre méltó, de jóval összetettebb témakör, hogy az ObjectARX.NET függvényeket hogy lehet LISP nyelvből meghívni. Megfelelő hibatűrésű C# kóddal a LISP nyelvű gyors segédprogram fejlesztés is támogatható lenne.

Hasznos lehetne az SQL adatbázis, Map objektum adatok, illetve JSON állományok kezelésének programozási kérdései, a szolgáltatók által készíttetett saját nyilvántartó és XML előállító programokkal való együttműködésben.

Látható, hogy ezen témakörből akár több lexikon kötet méretű anyagot lehetne írni. Csak remélni lehet, hogy a segédlet rövid mérete ellenére hasznosnak bizonyul a mérnök kollégák számára, és esetleg az általuk elkészített programok valamilyen módon – akár valamelyik szakági szervezet által fenntartott nyilvános fórumon – megosztásra kerül a mérnöktársadalommal.

9. Irodalomjegyzék

[1] Nemzeti Média- és Hírközlési Hatóság Egységes Hír-Közmű
https://nmhh.hu/hir-kozmu

[2] Nemzeti Média- és Hírközlési Hatóság Egységes Szakági Tervezéstámogató Rendszer (ESZTER) https://nmhh.hu/eszter

[3] AutoCAD Map 3D referencia és oktatóanyagok
https://aps.autodesk.com/developer/overview/autocad-map-3d

[4] MapGuide API Referencia
https://mapguide.ca/mapguide/help/webapi/index.htm

[5] Infrastructure Modelling DevBlog
https://adndevblog.typepad.com/infrastructure/

[6] AEC DevBlog
https://adndevblog.typepad.com/

10. Mellékletek

Állománynév	Állomány rendeltetése
MMK_plugin.zip	Az AutoCAD Map 3D kiegészítő Visual Studio projekt könyvtár tömörítve, a plugin forráskódjával.

A sorozat keretében eddig megjelent kiadványok

2017.-2022.

A korábbi kiadványokat keresse a kamara weboldalán (www.mmk.hu), a FAP pályaműveket tartalmazó menüben.

2023.

101.	SZILÁGYI Zsombor Dr.,	Az energiahordozók jövője
102.	BLAZSOVSZKY László	Szakmai útmutató felhasználási helyek gázellátását biztosító elosztóvezeték leágazásának létesítéséhez
103.	ÁKOSHEGYI György Dr., BORBÉLY Tibor, DIÓS András, EÖRDÖGH Zsolt	Medencés fürdők vízgépészeti tereinek tervezési szempontjai
104.	KISS Jenő dr., METZING Ferenc dr., CSERMELY Gábor, dr. HEGYI Dezső, KÖNCZÖL Gyula, DEZSŐ Zsigmond	Szakmai Útmutató az épületek, épületszerkezetek bontásához
105.	MÓGA István Dr.	Atomerőművi alapok
106.	BAK Edina, FEJES Gábor, HONTI Imre, HUDACSEK Péter, SCHELL Péter	Talajmechanikai laboratóriumi vizsgálatok, azok megtervezése és eredményeinek felhasználása a geotechnikai tervezői gyakorlatban
107.	EGRI Sándor, BUZÁS Zoltán	AutoCAD alapon készített tervdokumentációkból EHO szerinti XML fájl készítése (objektumsablon készítése)
108.	BONDOR Gabriella, CSORDÁS Szilveszter, KANOSIK Ilona, PÓLYA Endre, VARJÚ József	Szakmai ismeretek a jogosultsági vizsgára
109.	KAKUK Ilona	Az Informatikai Tervező Szakmai Útmutatója (109./1-2-3.)
110.	GIORIS Nikolaos, MENYHÁRT István Zsolt, OLÁH Róbert, TAKÁCS Bence dr., VASS Imre	Digitális mellékletek készítése az M.2 (2021.) mérnökgeodéziai tervezési segédlethez
111.	ZSIDAI László Dr., SARANKÓ Ádám Dr., SZABÓ Péter	Az additív technológiák terméktervezési és technológiai sajátosságai
112.	REINIGER Róbert, BARNA Sándor, BITE PÁLNÉ DR. PÁLFFY Mária, MIHICS Dalma, PINTÉR István	A Li-ion alapú akkumulátor, illetve akkumulátor részegység gyártás környezetvédelmi hatósági engedélyezésének környezetvédelmi alapkövetelményei - Szakmai segédlet környezetvédelmi szakértők, illetve hatósági eljárási szereplők részére
113.	TOKODY Dániel Dr., SCHOTTNER Károly, HADDAD Richárd, ADY László, ÁGOSTON Gergő, DRABIK Gergő, HAN Xiaoping, CSAPÓ Dániel, SZŰCS Marcell, FELHŐS Dávid Dr.	Napelemes rendszerekkel együttműködő energiatárolók létesítése
114.	GYIMESI András Dr., BOHÁCS Gábor Dr., SÜLLE Miklós, GÓDOR Balázs	Építőgépész mesteriskola tantárgyi felépítés és a hozzá kapcsolódó tematika kidolgozása
115.	BOROS János	Az atomerőművi rendszerek és rendszerelemek tervezésénél figyelembe vett elvek összefoglalása
116.	PRIMUSZ Péter Dr.	Hajlékony útpályaszerkezetek szükséges erősítőrétegének meghatározása roncsolásmentes teherbírásmérés alapján

117.	EGRI Sándor, BUZÁS Zoltán	AutoCAD alapon készített tervdokumentációkból EHO szerinti XML fájl készítése (objektumsablon készítése)

2024.

118.	SZABÓ Kinga Dr., ÓMAJTÉNYI András, TÓTH Péter, ZSIGMONDI András	A beruházáslebonyolító tevékenysége - Szakmai ajánlás a beruházáslebonyolító fogalmának meghatározására, felelősségére és jogosultságának szabályozására
119.	SZILÁGYI Zsombor Dr.,	Feladatok a klímacélok teljesítéséhez
120.	BLAZSOVSZKY László	A csatlakozóvezetékek és a felhasználói berendezések létesítésével kapcsolatos gyakorlat, és a változtatás szükségességének elemzése
121.	PÓLYA Endre, BALOGH Attila, ERDEI Tímea, VARJÚ József	Műszaki segédlet az orvostechnikai eszközpark üzemeltetéshez - Tervezési segédlet, szakmai útmutató, tananyag az MMK kötelező szakmai továbbképzéshez, valamint dokumentum egészségügyi létesítménymenedzsment és minőségmenedzsment részére.
122.	BOROS János, CSALLÓKÖZI Zoltán, DOBÓ István, HUNYADI Sándor, KÁLLAI-BORIK Róbert, PODONYI Gábor, SZITA Gábor, DR. SZÜCS Botond, PERGERNÉ NAGY Edit, GYŐRI Csaba	Magyarország villamosenergia ellátásának lehetőségei
123.	DR. TAKÁCS Bence, HRUTKA Bence Péter, TAKÁCS Regina	Szakmai útmutató vonalas létesítmények 3D modellezéséhez (geodéziai tervezők részére)
124.	LIPOVICS Tamás, SZENDEFY János, VÁSÁRHELYI Balázs	Kockázatok elemzése a mélyépítésben
125.	KAKUKK Ilona Klára, NÓGRÁDI Gábor	Informatikai Tervező Szakmai Útmutató
126.	EGRI Sándor	Az NMHH ESZTER aktuális verziójának használata során feltárt hiányosságok, problémák kezelése az AutoCAD programozási eszköztárával
127.	DR HANCZ Gabriella, DR. ENGI Zsuzsanna, JANCSÓ Béla	A kék-zöld infrastruktúra elemek tervezési irányelvei
128.	HAJNAL János, DR. KOREN Csaba, LITKEI Bálint, VÁLÓCZI Dénes György	A Közúti biztonsági auditjelentések szerepe az önkormányzatok pályázati gyakorlatában
129.	HÓZ Erzsébet, BORBOLÁNÉ KOVÁCS Gabriella, DR. MILETICS Dániel, PÁL Péter, VÁLÓCZI Dénes, VEDRŐDI Tamás	Személyi sérüléses közúti közlekedési balestek adatgyűjtési módszertana
130.	SZOMOLÁNYI Tiborné, LAKATOS István, LUKÁCS Tamás	Segédlet a megvalósulási dokumentációk készítéséhez, használatbavételi, fennmaradási és bontási engedélyek dokumentációinak elkészítési segédlete
131.	KATONA Gábor, FÜZÉR Ferenc, MOLNÁR Béla	Hírközlési hálózatok építési technológiai gyűjtemény - Bontások, útátfúrások, anyag és munkaigények összefoglaló tervezői anyag
132.	KOVÁCS József, NAGY Benedek, SZABÓ László, DR. SZEPESHÁZI Attila	Geotechnikai és tartószerkezeti tervező együttműködése talaj és felszerkezet egymásra hatásának vizsgálata esetén

133.	MÁRKUS Miklós, MUNTAG András	Ipari zajmodell adatszolgáltatási segédlet - Segédlet szakági tervezők részére környezeti zajmodell készítéséhez