

# Tesztelés

## (Meteorológia)

### Tesztelés felosztása

Kezdjük azzal, hogy mely class-okat, milyen módszerekkel lehet tesztelni (illetve milyen szempontok alapján kéne). A projekt 2 részre lett osztva (3 ha számoljuk a tesztet), a „ClassLibrary” ami az adattárolásra fókuszál, és a „Systems” ami a felhasználóhoz kapcsolódó osztályokat tartalmazza.

Az adattárolással kapcsolatos osztályokat nyugodtan lehetett Unit tesztekkel tesztelni, de mivel a felhasználó (és neki metódusai) mind a konzollal interaktálnak, így azokat kézzel kellett ellenőrizni. Emelet az is a feladat leírás része volt, hogy a rendszerünknek képesnek kell lennie több ezer elemet is kezelni úgyhogy arra a teljesítményt is mérni kell majd (Ez egyébként nem volt a legegyszerűbb, mivel csak egy 4 node-ból álló példa input volt kiadva, így hiába mondta a feladat, hogy nem kell, de meg kellett csinálnom az exportálást, hogy lehessen arra is hatalmas bemeneti file-t rakni)

### Unit tesztek

A unit tesztek az adattárolás két fő tagja alapján lettek ketté osztva. A DataNode tesztek, amik ugye 1-1 adattag helyes működését vizsgálták, és a MeasurementSystem ami pedig a teljes adathalmaz tárolásáért felel.

### DataNode tesztek

A DataNode minden leszármazottjához az alábbi 3 teszt tartozik:

1. A gép képes létrehozni az adott fajta adat node-ot. Ezt annyival ellenőrzöm, hogy a létrehozott instance neve az elvárt-e (nincs túlbonyolítva)
2. Létrehozunk az adott node fajtából egyet az alapértelmezett mértékegységgel, és utána megvizsgáljuk, hogy átváltva az ismert mértékegységekbe helyes adatot ad-e vissza.
3. Az előzőt próbáljuk, csak fordítva -> Megpróbáljuk a node-nak utólag egyesével az adott ismert mértékegységekben mérve beállítani az értékét, majd megnézzük végén hogy a várt értéket kapták-e.

Ezeket mind a 4 jelenlegi adat típuson teszteljük.

A legvégén pedig van még egy extra teszt. Ez azt vizsgálja meg, hogy a DataNode statikus „IsUnitSupportedForType” függvényé helyesen működik-e. Ez ugye később a DataNode-ok dinamikus létrehozáshoz fontos.

### MeasurementSystem tesztek

Ezek is két részre lettek vágva. A Basic és a Generator tesztek.

1. Első teszt lényege, hogy képes-e olvasni. Megvizsgáljuk, hogy elég adat van-e a rendszerben a teszt végére.
2. Egy tökéletes felépítésű file helyesen van-e beolvasva. Itt ellenőrzésnek pár konkrét értéket megkeresünk, és azokat ellenőrizzük.
3. Itt egy nagyobb, de még mindig tökéletesen formázott file sikeres beolvasása a cél. Itt a DataNode-okat fajtára osztva megszámloljuk, és azt ellenőrizzük.
4. Egy olyan file-t próbálunk beolvasni, aminek a sortörése nem az eddigi megszokott módon van, és azt is be tudja-e olvasni.

5. Egy rosszul megformázott file (sok hiányos adat) sikeresen van-e beolvasva. Azaz az egyetlen jól szereplő adattag benne be lett-e olvasva, és a program lefutott-e (leállítás nélkül, ami ugye elvárás).
6. Egy több ezer adattagból álló file beolvasása (Ez exportálással készült). Ez főleg a performance mérésre van.
7. De aztán rájöttem, hogy ez nem az igazi, lehetne ennél nagyobb is. Egy 35 000+ elemből álló file beolvasása a hetedik. Itt persze az egyetlen ellenőrzés, ami belefér az a darabszám ellenőrzése (Kézzel nem tudom, hogy írhatnék ennél pontosabb ellenőrzést)

Itt váltunk a Generátorok tesztelésére

8. Egy napnyi adatot generálunk. Leellenőrizzük, hogy az adatok mennyisége annyi-e, amennyire számítunk, és hogy azok mind ott vannak, ahol számítunk rájuk.
9. Teljesítmény teszt a generátornak. Mind a négy eddig létrehozott adat kategóriában generálunk 1-1 évnyi adatot. Darabszámukat ellenőrizzük (35 000+).
10. A generálás betartja a megadott szabályokat. Az előzőt megismételjük, de most csak akkor számoljuk az elemeket, ha a megadott kereteken belül vannak.

### Unit tesztek eredménye:

Test Explorer

Test run finished: 24 Tests (24 Passed, 0 Failed, 0 Skipped) run in 375 ms

Search (Ctrl+I)

0 Warnings 0 Errors

Test	Duration	Traits	Error Message
✓ Tester (24)	283 ms		
✓ Tester (24)	283 ms		
✓ DataNodeTests (14)	5 ms		
✓ 01. Tester is working	< 1 ms		
✓ 02. Creating Pressure Node	< 1 ms		
✓ 03. Converting from base hPa t...	< 1 ms		
✓ 04. Converting from custom uni...	< 1 ms		
✓ 05. Creating Temperature Node	< 1 ms		
✓ 06. Converting from base Kelvi...	< 1 ms		
✓ 07. Converting from custom uni...	< 1 ms		
✓ 08. Creating Humidity Node	< 1 ms		
✓ 09. Converting from base % to...	3 ms		
✓ 10. Converting from custom uni...	< 1 ms		
✓ 11. Creating WindSpeed Node	< 1 ms		
✓ 12. Converting from base m/s t...	2 ms		
✓ 13. Converting from custom uni...	< 1 ms		
✓ 14. Unit checking	< 1 ms		
✓ MeasurementSystemTests (10)	278 ms		
✓ 01. Can Read	< 1 ms		
✓ 02. Perfectly Formated File was...	3 ms		
✓ 03. Perfectly Formated Bigger Fi...	< 1 ms		
✓ 04. Test on non-one-line-one-n...	1 ms		
✓ 05. Bad file read correctly	18 ms		
✓ 06. Huge File read test	16 ms		
✓ 07. Giga File read test (MBs)	152 ms		
✓ 08. Generator basic test	< 1 ms		
✓ 09. Generator Huge test	44 ms		
✓ 10. Generator worked in bounds.	44 ms		

Run | Debug | Profile

Group Summary

Tester

Tests in group: 24

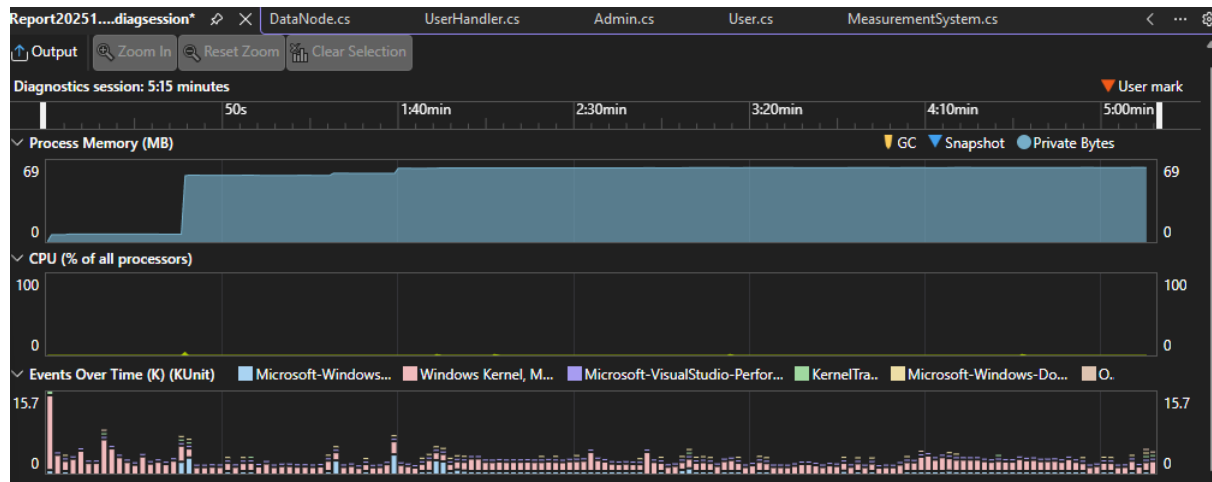
Total Duration: 283 ms

Outcomes

✓ 24 Passed

## Performance Tesztek:

Már ugye a unit tesztek között is van aminek az a célja, hogy mutasson egy nagyjából futásidőt hatalmas adatmennyiségek mellett, de ennél pontosabb mérést is végeztem. Az alábbi diagramm egy olyan futtatásból van, ahol minden egyes felhasználói lehetőségen végig mentem (az admin-on is). Betöltöttem a 35 000 elemű file-omat, majd generáltam hozzá még fél évnyi nyomás adatot, majd exportáltam az egészet, és utána végig mentem a többi felhasználói lépésen is.



Ahogy látható kifizetődött a CPU orientált tervezés, a CPU szinte meg sem érezte a futtatást (a CPU-m egy i5-12400F, de ez akkor is szép, hogy ő gond nélkül elbírt 40 000+ elemet).

