Tuska Balázs

Valami Commander Felhasználói és Fejlesztői Dokumentáció

Témavezető: Porkoláb Zoltán

ELTE IK
2004

Tartalomjegyzék

1. Bevezető

az igény mindig is megvolt az emberekben, hogy a gépüket úgy használhassák, hogy az kényelmes legyen számukra. Alapértelmezésben az operációs rendszerek kevéssé mondhatóak kényelmesnek vagy éppen átláthatónak. Bár a modern rendszerek alapból kínálnak több lehetőséget, hogy a felhasználók életét megkönnyítsék, de sokak számára ez nem megfelelő, így sok esetben a rendszer alkotói több hátrányt csinálnak, mint előnyt. Sok esetben nem is szükséges a legtöbb embernek általános számítógép használat közben, hogy mélyebben belelásson, átlássa a rendszert, de sok szempontból számukra is előnyös lehet, bár használata nem merül fel bennük, mivel a lehetőségéről sem tudnak. A haladóbb felhasználók viszont, el sem tudják képzelni, hogy ne lássák át a rendszert olyan mértékben, amennyire az számukra lehetséges, az ilyen ember szereti tudni mi hol van a gépén, szeret rendetrakni.
Az én programom ennek a rendnek a megteremtésében és a rendszer átláthatóságában nyújthat nagy segítséget azzal, hogy az ember mindig láthatja, hogy éppen mit csinál, és láthatja egyből az eredményét is.

1.1. A programról

Ez a program egy olyan file manager, mely követve a hasonló témájú programok hagyományait, két paneles. Ezen panelok között tudunk különféle műveleteket végezni, főleg file-műveleteket, például másolás, mozgatás, törlés stb. (lásd az ezzel foglalkozó fejezetet, bővebb információért). A program feltett szándéka, hogy a számítógép felhasználóinak az életét megkönnyítse, ha azok általános file-műveleteket akarnak végezni. Ezen felül képes FTP használatára, kezeli a legtöbb tömörített állományt (bizonyos feltételek mellett), tetszőleges állomány tartalmát megjeleníti, könyvtárak méretét is megnézi, kezeli a szimbolikus linkeket, kezeli a jogokkal kapcsolatos dolgokat, és általában mindent megcsinál ami file kezeléssel kapcsolatos vagy ami hozzá tágabb környezetben megtalálható. Például a programmal nem lehet file-t szerkeszteni, de lehetőség van, hogy a program egy másik külső programot hívjon meg, ami már képes szerkeszteni azt.
A program Linux X környezetben fut, eddig annak minden terjesztése alatt. (Bővebb információ a telepítésnél.) A program licensze igazodik a free környezethez: GPL. E miatt szabadon terjeszthető mind a bináris, mind a forráskód is, sőt a forrás szabadon módosítható,

és a módosítás is szabadon terjeszthető.

1.1.1. Motiváció és történet

A program megírásához az alábbi út vezetett: Régebben a 90-es évek elején és közepén, mikor az átlag felhasználó számára az operációs rendszert a DOS jelentette, mely az embert nem kényeztette el túlságosan, már rendelkezésre állt több olyan file kezelő melyek, a mai, így az én programom alapját is képezte. Ezek természetesen még szöveges üzemmódban működtek. Minden file-manager szülőatyja a Norton Commander, mely abban az időben etalonnak számított, minden művelet elvégezhető volt vele az akkori igényeknek megfelelően. Ez a szoftver fizetőssége ellenére igen elterjedté, egyeduralkodóvá vált, de felmerült egy ingyenes változatának a terve. Így megszületett a Volkov Commander és a Dos Navigator, mely utóbbi igencsak testre szabható volt, de az új operációs rendszer, a Windows 95 megjelenésével elkezdte kiszorítani a szöveges file kezelőket. Így azoknak igencsak leáldozott, ez nem jelentette persze a teljes halálukat, mert ezek többsége megjelent olyan új verziókkal, melyek fel lettek készítve az új rendszer újdonságaira, például a hosszú állománynevek kezelése. Érdekesség, hogy ebben az időben jelent meg a FAR Manager, mely csak Windows alatt futott, ennek ellenére mégis szöveges üzemmódú volt.
Az új Windows új lehetőségeket is hordozott. Az Intéző a Windows beépített file-kezelőjének lehet felfogni, amit egy átlag felhasználó meglepően jól képes használni, de sokaknak (közöttük én is) az igényeit nem bírta kielégíteni, ezért a Windowsos API-t felhasználva megjelent a Windows Commander, későbbi nevén a Total Commander. Ez a file-kezelő verzióról verzióra növekedve egyre több funkcióval szolgált, és szinte minden igényét kielégítette, akár a legfinnyásabb embereknek is. Sokat dob rajta, hogy külső plugineket lehet beilleszteni a programba, ezzel is növelve a tudását.
Bennem már a DOS-os időkben felmerült egy saját Commandernek az ötlete. Ezt el is kezdtem írni, akkoriban még Pascal nyelven. Ezt egy darabig írtam, de semmi különlegességgel nem rendelkezett sőt butább volt mint az addigiak, a fejlesztés lassú volt, és különféle nehézségekbe ütköztem, ezért a fejlesztés leállt. Itt hiányzott a kellő motiváció, mert a meglévő programokat nehéz volt túlszárnyalni, csak érdeklődés szintjén csináltam a dolgot, gyakoroltam a programozást.
Az ELTE-n folytatott tanulmányaim során megismerkedtem közelebbről a Linux operációs rendszerrel, eddig ezt csak hobbi szinten néztem. Most már ott tartok, hogy nincs Windows a

gépemen 100%-ig Linuxot használok. Az új rendszerrel való ismerkedésem során természetesen kerestem a DOS/Windowshoz hasonló programokat, hogy ezzel is könnyítsem a beilleszkedésemet. Egyből meg is találtam a Midnight Commander nevű szöveges üzemmódú file-managert, ami egyedülálló a Linux programok között. Teljes mértékben illeszkedik a magához a rendszerhez és a filozófiájához. De én, mint desktop user többet szerettem volna, egy olyan programot, ami X alatt grafikusan tudja azt, vagy közel hasonlót, mint az MC. Persze az MC-t is lehet használni a grafikus felület alatt, de akkor is csak szöveges marad. Keresgéltem az Interneten, és találtam is rengeteg programot. Ebben a témában az egyik etalonnak számító program a Gnome Commander, kisebb nehézségek árán sikerült beüzemelni a legfrissebb verzióját. Persze a használat során rengeteg hiányosságra és bug-ra fény derült, így majdhogynem használni sem lehetett, de hosszabb távon semmiképpen sem. Emellett még ott volt a többi rengeteg kezdeményezés, de egyik sem bizonyult túl használhatónak. Talán a Krusader nevezetű file-kezelőt mondanám a legjobbnak, mert ez kezelt tömörített állományokat, és ez volt az egyetlen, ami képes volt scp-zni. Ennek ellenére ez a program sem nyerte meg a tetszésemet, mert több kényelmi szempontnak nem felelt meg. Így grafikus file-kezelő híján úgy döntöttem, hogy ismét kísérletet teszek egy megírására.

Az első készületeket tavaly nyáron tettem meg, de elég sok akadályba ütköztem. Fejlesztő eszköznek, ELTE-s hagyományoktól eltérően nem a Kdevelop és Qt Designer párosát választottam hanem a Borland cégnek a Kylix 3.0-ás eszközét. Sokan tévesen az hiszik, hogy ez egy Delphi átirat Linuxra, de tévednek, mert a a 3.0-ás verzió óta van Pascalos és C++-os része is. Ez az eszköz kinézetre teljesen a Windowsos Delphire illetve Cbuilderre hasonlít, de azoktól gyökeresen eltérő háttérrel rendelkeznek. Tehát az én választásom a Kylix for C++-ra esett. Az Interneten számtalan file-kezelő kezdeményezés található, ezek közül amik Kylixban lettek fejlesztve azok mind a Pascal részét használták, de a függvénynevek azonosak voltak. A programoknak, kivétel nélkül, megtalálható volt a forráskódja is a Neten, ezzel ötleteket adva és segítve a fejlesztést. Komolyabban a fejlesztéssel 2004 januárjában kezdtem foglalkozni, sorra jelentek meg az újabb verziók, egyre használhatóbbak és stabilabbak. Áprilisra sikerült elérni a stabilnak kimondott 0.4.0-ás verziót, ami majdnem teljesen megegyezik a jelenleg tárgyalt 0.4.1-essel. A program neve kezdetben File Commander volt (még DOS-os időkben hívtam így és ide is átvettem), de több sérelmezés után átneveztem, Mivel senki nem tudott értelmes nevet kitalálni neki, ezért meghatározatlan időre Valami Commandernek kereszteltem el.

1.1.2. Mire használható?

Ennyi bevezető után feltehetjük a kérdést, hogy mire is jó ez a program, kiknek érdemes alkalmazni. Nos a válasz egyszerű és nem is egyszerű. Röviden megfogalmazva a Valami Commander egy olyan két-paneles file-kezelő, amely általános file műveletek végzésére alkalmas, ezek a file-ok elhelyezkedhetnek a merevlemez valamelyik particiójának könyvtárában, tömörített állományban (EXTFS), vagy egy FTP szerveren. A file műveletek alatt is sok mindent lehet érteni: másolás, átnevezés, többes átnevezés, törlés, jogok kezelése, file tartalmának megtekintése, szimbolikus linkek kezelése, file illetve könyvtár létrehozása, file-ok szerkesztése stb.

1.1.3. Mire nem használható?

Ez a program rengeteg dologra nem használható, ez főleg köszönhető a még kezdeti stádiumnak és a megfelelő tudás hiányának. Jelenleg ismert legnagyobb hiányosságok: drag'n'drop hiánya mind programon belül, mind más programokkal kooperálva, scp-zés lehetőségének hiánya, normális nézegető hiánya (bár külső nézegetővel kompenzálható), egységes VFS kezelés (bár ennek nem sok hátránya van a felhasználó fele, inkább fejlesztés hátráltató). Emellett számos olyan dologra nem alkalmas, melynek nem sok köze van a file-ok kezeléséhez.

1.2. Telepítés

A program az Interneten elérhető már előre lefordított binárisként és forráskódban is.

1.2.1. Követelmények és Ajánlások

Természetes alapkövetelmény a Linux operációs rendszer megléte X window grafikus felülettel. Ezen belül a terjesztésre és ablakkezelőre nincs külön megkötés. (Eddig tesztelve van: SuSE, RedHat, Fedora Core, Debian, Mandrake, Uhu Linuxok legújabb verziói alatt (2.4.x és 2.6.x-es kernelekkel egyaránt), és a BlackBox, IceWm, Gnome, KDE ablakkezelők legújabb verziói alatt). Ezek megléte mellett az összes függőségnek teljesülnie kell, de a

program alapműködése mindenféleképpen garantált (ha nem így lenne az kérjük jelezze e-mailben).
Emellett a program összes funkciójának elérése érdekében szükség lehet a különböző tömörítő programok meglétére (bzip2, gzip, tar, rar, ezek általában részei a terjesztésnek). Valamint alapértelmezetten a mplayer médialejátszóra, kghostview pdf- és ps nézegetőre, valamint a kompare nevű programra a fileok össze hasonlításához. Ezek legtöbbje része szokott lenni a különböző disztribúcióknak.

1.2.2. A telepítés menete

A terjesztéshez az RPM formátumot választottam, mert ennek elkészítése során külön létrehozza a bináris telepítőt és külön egy forráskód csomagot is. Manapság nagyon sok Linux terjesztés RPM alapú, de a legtöbb terjesztés, ami nem is RPM alapú az is támogatja ezt a csomagformátumot.
Az RPM lehetőséget biztosít, hogy installálás közben a programhoz szükséges függőségeket leellenőrzi, emellett jó a verzió kezelője. Bár igazából csak root jogokkal lehet telepíteni, de tetszőleges felhasználó ki tudja csomagolni az állományt, így nem szükséges a használathoz root jog.
A telepítést az alábbi utasítással lehet elvégezni:

```
rpm -i vc-current.i586.rpm
```

Ha már van fen a gépen egy korábbi verzió, akkor a következő utasítás segít:

```
rpm -u vc-current.i586.rpm
```

Ha installálás alatt függőségi problémákat jelez, akkor vagy teljesítjük a függőségeket, vagy az alábbi utasitás szerint járunk el (a nem RPM alapú terjesztéseknél lehet olyan probléma, hogy az RPM adatbázisban nincsenek bejegyezve a telepített csomagok ezért jelez hibát annak ellenére, hogy a csomag fennt lehet):

```
rpm -i –force –nodeps vc-current.i586.rpm
```

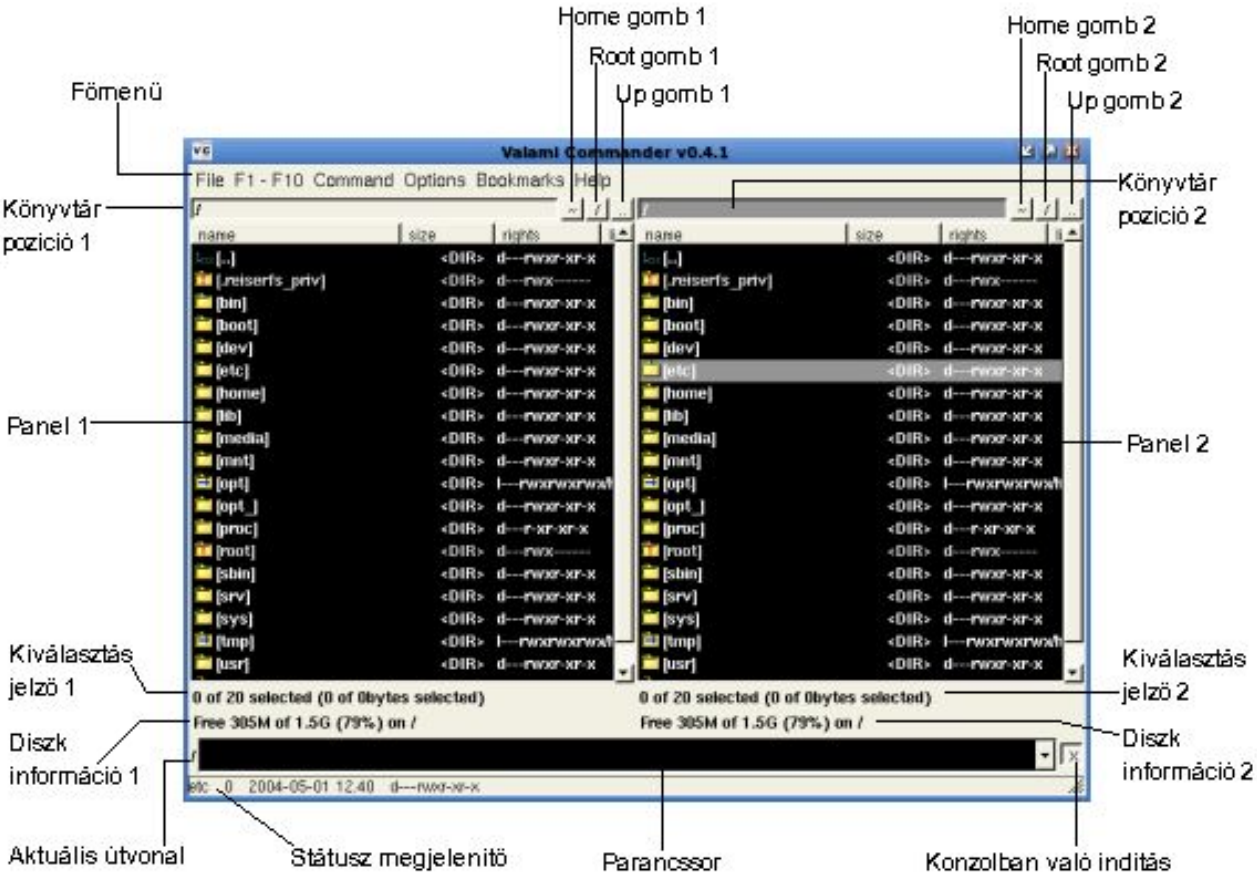
Installálás után a programot, minden felhasználó tudja majd használni a „vc” paranccsal, vagy a honlapról letölthető desktop-file-lal.

2. Funkciók részletesen a felhasználók számára

Ebben a részben felhasználói dokumentációhoz hasonlóan mutatom be a programot és annak elérhető funkcióit. Ez általában abból fog állni, hogy végig megyünk az egyes menüpontokon, mert ez nagyjából le is fedi a program tudását kis túlzásokkal. Természetesen kitérek az olyan megoldásokra is, melyek menüből nem érhetőek el, azt is leírom, hogy miért is nem.
Aki az egyes funkciók függvényeire és azok megvalósításaira kíváncsiak, azok lépjenek a következő nagy fejezetre, a fejlesztői dokumentáció részhez.

2.1. A fő ablak

A főablak az az, amit a program elindításakor alapból lát az ember. Természetesen ez az ablak lesz minden eseménynek a fő helyszíne, a többi ablak és a funkciók kiindulási helye. Ez az ablak számtalan sok elemet tartalmaz, melyet a következő ábrán szemléltetek:



Egyből szembetűnik az ablak kettéosztottsága, a két panellel és a hozzájuk tartozó információkkal. Most vegyük szemügyre a főbb komponenseket.

2.1.1. Panelek

A „Commanderek” általános jellemzője, hogy két panelt tartalmaznak, melyek mindig a hozzájuk kapcsolódó könyvtárak tartalmát jelenítik meg valamely szabályok szerint. Jelen esetben meg tudjuk választani, hogy az elemek rendezettsége milyen szempont alapján történjen meg: név, méret és idő, valamint növekvő vagy csökkenő sorrendben. A listához szorosan kapcsolódik a Scrollbar, ami akkor lesz aktív, amikor a listában túl sok elem van és nem fér el egyszerre a megjelenítőn.

A lista rendezettségét a tetején található oszlop azonosítókkal határozhatjuk meg. Például ha a „name” oszlop fejlécére klikkelünk, akkor név szerint lesz rendezve növekvő sorrendben, de ha már név szerint volt rendezve, akkor a rendezés irányát megfordítja. A Rendezést még menüből is be lehet állítani.

Természetesen a fejléccel az oszlopok szélességét is meg tudjuk határozni. Ha az oszlopok tartalát meg akarjuk változtatni, akkor az a beállításoknál tehetjük meg (ls. később).
A panelhez szorosan kacsolódik az a mező mely azt mutatja meg, hogy a panel éppen melyik könyvtár tartalmát jeleníti meg. Ez nagyjából nem bír különösebb funkcióval, de lehetőség van kijelölni azt, ezzel elmenteni a vágólapra. A könyvtár név mellett található három gomb, az egérrel való navigálást könnyíti meg. Lehetőség van a felhasználó „home” könyvtárába, vagy a gyökérbe, vagy egy szinttel feljebb lépni.
A lista alatt találunk információt a listában szereplő és a közülük kijelölt elemekről, megadva a számukat és a méretük összegét. Továbbá az aktuális könyvtár helyzetéről, hogy melyik partición helyezkedik el, és az mekkora méretű és abból mennyi szabad.
Ez a két panel határozza meg a navigációt is. Itt tudunk mozogni a kurzorral, kijelölhetünk elemeket, azaz meghatározhatjuk, hogy mely file-okkal kívánunk műveletet végezni. A panel lista tetején mindig a '..' áll, utána következnek a megfelelő sorrendben a könyvtárak, végül a megfelelő sorrendben a file-ok. (A nem létező helyre mutató link nem-determinisztikusan, néha könyvtár tulajdonságú, néha file.)

2.1.2. A parancssor

A parancssor a következő állandó kelléke a commandereknek. Ez rész mutatja meg az éppen aktuális könyvtárat, valamint innen tudunk indítani tetszőleges programot tetszőleges paraméterekkel. Megtalálható itt egy plusz gomb is („X”) a Windowsos és a Linuxos szöveges commanderekhez képest. Ez arra szolgál, hogy megmondja, hogy a programot a meghatározott xshellben indítsa-e el, vagy simán. Ez azért szükséges, mert egyfelől a szöveges programoknak kell egy shell, amibe írhatnak, másfelől egyes programok igénylik és nélkülük el sem indulnak, ilyen az mplayer is. Természetesen a legtöbb grafikus program nem igényel saját shellt.
Jellemző a parancssor és a panelek kapcsolatára, hogy ha egy betű lenyomása kihatással lehet a parancssorra még ha az nem is aktív. Ugyanígy ha a parancssor aktív és például a felfele nyilat nyomom meg akkor az a panelre lesz kihatással. De értelem szerűen az olyan gombok melyek mindkét félnél használhatóak (home, end, jobbra, balra) az a megnyomás pillanatában az éppen aktív elemre lesz hatással. Ez esettől eltérő lehet az enter billentyű hatása, mert ez aktív panel mellett is parancssoros hatást válthat ki, ha a parancssor mezője nem üres!

A parancssor rendelkezik történettel (history), mely a programból való kilépéssel automatikusan elmentődik, és következő alkalommal újból betöltődik. A történetbe azok az utasítások kerülnek be, akiket a parancssorba beírtunk, majd enterrel aktiváltuk őket. A történetet a legördülő menüre kattintva, vagy a CTRL+LE/FEL billentyűkombinációval érhetjük el.

2.1.3. A statusbar

A statusbar, azaz a program legalsó része, különböző hasznos információk megjelenítésére van. Például ha egy file neve túl hosszú, akkor a kurzort fölé pozicionálhatjuk, és megláthatjuk a teljes nevét, sőt a többi tulajdonságát is. Továbbá a jelenlegi verzióban megjeleníti az összes indított programot és azok paramétereit. Ez azért lehet hasznos, mert a felhasználó nem feltétlenül tudja mi zajlik a háttérben, és ha valami nem megy, akkor ez segíthet a probléma megoldásában. Azaz ha például a felhasználó egy file-t akar szerkeszteni, de az alapértelmezett szerkesztő (kwrite) nincs a gépén, akkor az nem indul el. Ekkor a felhasználó nem sejtheti, hogy mi van, de a statusbarra pillantva látja, hogy a program a „kwrite file_neve” utasítással próbálkozott.

2.1.4. A menü

A menü az mondhatni a műveletek közötti eligazodást segíti. A gyakorlott felhasználónak ritkán kell használnia a menüt, mert az összes menüponthoz gyorsgombok (hotkey, shortcut) rendelhetők, ezzel is segítve a gyorsabb használatot. Mondhatnám, hogy ami a menüben nincs benne az nincs is menne a programban, persze ez nem teljesen igaz, de a főbb funkciók mindegyike elérhető a főmenüből. A menü gyorsbillentyűje alapértelmezésben az F9.

A File menüpont

Ez az elő menüpont. Túlságosan sok dolgot nem tartalmaz, de itt nyílik lehetőség a panelek rendezettségeinek megváltoztatására. Ez történhet név, dátum valamint méret szerint, csökkenő és növekvő sorrendben egyaránt. Ha megváltoztatjuk a rendezést, akkor a változás mindig csak az aktív panelre lesz érvényes. Az éppen aktuális rendezési módot és irányt egy-egy pipa jelzi, a megfelelő sor bal oldalán.

Az F1-F10 menüpont

Ez a menüpont onnan kapta a nevét, hogy az alapértelmezésben az F1-től F10-ig lévő billentyűkre állított funkciók itt találhatóak. Ezeket a műveleteket hívnám én a Commanderek alapműveleteinek: nézés, szerkesztés, másolás, mozgatás, könyvtár létrehozása, törlés, főmenü, kilépés. Ezenfelül plusz a következők: nézés külső programmal, új file létrehozása, helyben másolás, átnevezés. A VC ezek közül az összeset tartalmazza, de a helyben másolást menüben nem (így a gyorsbillentyűje sem változtatható meg). Az egyes funkciók leírása a file műveletek résznél található részletesen.

A Command menüpont

Ebben a részben találhatóak a különlegesebb és nem mind nap használt funkciók, esetlegesen azok a műveletek melyek Linux specifikusak. Ezek: szimbolikus linkek kezelése (létrehozás, szerkesztés), jogokkal kapcsolatos dolgok (jogok, tulajdonos állítás), file keresés, file tartalom összehasonlítás, paneltartalom frissítés, könyvárak méretének meghatározása, csoportos átnevezés, kijelölés (invertálás, invertálás maszk szerint, kijelölés megszüntetése, összes kijelölése), ftp kapcsolat (új kapcsolat, meglévő megszüntetése).

Az Options menüpont

Természetesen minden program alapvető lehetősége, hogy testre szabható legyen. Ebben a részben erre nyílik lehetőség. Lehet részletesen beállítani, majd elmenteni az adott beállításokat. Haladóbb felhasználók akár magát a konfigurációs file-t is módosíthatják, melynek az érvénybelépéséhez nem kell kilépni a programból majd újra elindítani azt, hanem elegendő ebben a menüpontban újraolvastatni. A VC is igazodik ahhoz a Linuxos konvencióhoz, hogy a „CTRL-+/-”-szal növelhetjük illetve csökkenthetjük a betűméretet.

Bookmark és Help menüpontok

A bookmark menüben találhatóak meg azok a könyvtárak melyeket a felhasználó gyorsan el szeretne érni. A menü elérése CTRL+D billentyűkombinációval is lehetséges. Ez a menü két

részből áll, melyeket egy vonal választ el. Az első rész tartalmazza a könyvtárakat, a második annak a lehetőségét, hogy az éppen aktuális könyvtárat felvegyük a listába (mert ez ugye dinamikus). A bookmark az új könyvtár hozzávételévek elmentődik a $HOME/.vc/bookmark file-ba (a konfigurációs file-ok helye a $HOME/.vc könyvtár).
A help rövid, angol nyelvű segítséget nyújt a felhasználónak, az alapvető funkciók leírásával. Emellett megtalálható itt az About box, ami a program nevet, verziószámát és a készítőjét adja meg.

Most nézzük a program egyes részeit részletesebben.

2.2. A Viewer

A Viewer magyarul a nézőke arra való, hogy a file-ok tartalmát megjelenítsük. A nézőke szorosan a program része, azaz nem külön állományban van. Alapértelmezésben „SHIFT-F3”-ra nézi meg a kurzor alatti file-t. Ha nincs definiálva a külső nézőke, akkor alapértelmezésben F3-ra is a beépített nézőkét használja. Továbbá menüből is elérhető. Előfordulhat, hogy az ember véletlenül könyvtáron nyom nézést, ekkor sincs túl nagy baj, mert a program ekkor a könyvtár méretét fogja kiszámolni (kiszámoláshoz belelép az összes alkönyvtárba és a file-ok méreteit adja össze, a szimbolikus linkeket nem fogja követni, a végtelen ciklusok elkerülése végett).
A beállítások között megadható külső nézőke is, ha valakinek nem felelne meg a beépített.
A belső nézőkéből az alábbi módokon lehet kilépni: bezáró gomb, ablakkezelő ablakbezáró gyorsbillentyűjével, 'Q', F3, F10, ESC.
A programban lehetőség van string keresésére. Ezt az ALT-F7 illetve F7 billentyűk valamelyikével érhetjük el. Ezután a feldobott ablakban egy szabályos maszkra lehet keresni (a maszkokról bővebben a róla szóló fejezetben), ez elsőre furcsa lehet, de sok lehetőséget is magában rejt. A keresés mindig az aktuális kurzor pozíciótól indul, és az első előfordulásig vagy file végéig keres. Ha nincs találat, akkor ugyanott marad, ahol keresés előtt is volt, különben arra a sorra ugrik, ahol a találat volt, és kijelöli az illeszkedő szöveget. Ha volt találat, akkor a következő ugyanolyan keresés indításához az 'N'-t kell megnyomni.
A nézőke a sortörés karaktert is figyelembe veszi, azaz egy EOL-ra, új sort fog kezdeni. Ezen felül a hosszú sorokat is törni fogja, 500 karakter a limit. Egyszerre csak a látható ablakrészben található karaktereknél háromszor többet olvas be egyszerre a nézőke. Ez azért

szükséges, mert a nagy file-ok egyszerre való megnyitása lehetetlen. Az 500-as limites sortörés is pont ezért kell, mert ha egy nagy file-ban nincs sortörés, akkor egyszerre be kellene olvasni az egészet, ami szintén nem lenne jó mind sebesség, mind memóriafoglalás szempontjából sem. Tehát éppen mindig csak annyit olvas be amennyi a megjelenítéshez szükséges, plusz egy kis cache. Ezért mindegy, hogy a file mekkora, a nézés mindig ugyanannyi időbe telik. Természetesen a keresés már függ a file méretétől.
A navigáció az eddigi billentyűkkel plusz a szokásos billentyűkkel zajlik: kurzorgombok, page up, page down, home, end. Ennek megfelelően lehetőség van soronként és oldalanként lapozni, valamit a file elejére illetve végére ugrani.

2.3. A Multi Rename Tool

Ez az eszköz több file átnevezésére alkalmas bizonyos szabályok szerint. A 0.4.1-es verzióban jelent meg a szükség miatt. Az átnevezés során egyből láthatjuk a fileneveken az eredményt anélkül, hogy valóban át kellene nevezni. Az átnevezés egy speciális formátummal határozhatjuk meg, melyet egy interpreter elemez le minden file-ra alkalmazva.
Az ablak egy formátum meghatározó mezőből, egy kimenetet mutató mezőből, az információs részből valamint az érvényesítő gombokból áll. Hatástalan kilépés ESC-vel, Cancel gombbal, ablakkezelő ablakbezáró billentyűkombinációjával lehetséges. Ha a beállított formátumot akarjuk érvényesíteni, akkor az Ok gombon klikkeljünk, vagy az enter billentyűt nyomjuk le.
Az információs rész tájékoztat a lehetséges függvényekről, ezeken kattintva megjelenik a megfelelő minta, ezzel is kímélve a felhasználót a sok gépeléstől, mert egyes esetekben igazan bonyolult fuggvenyekkel lehet kihozni, amit szeretnénk.
A lehetséges függvények:

```
- %FULLNAME             - a file teljes neve
- %NAME                 - a file neve kiterjesztés nélkül
- %EXT                  - a kiterjesztés, mely tartalmazza a pontot is
- %TOUP<expr>           - az <expr>-t átalakítja nagybetűssé
- %TOLOW<expr>          - az <expr>-t átalakítja kisbetűssé
- %STRING{char*}        - char*-gal megadott stringgel tér vissza
- %POS<expr1><expr2>    - az <extr2> első előfordulási pozíciójával tér
                          vissza <expr1>-ben
```

```
- %ACCLESS<expr>        - az <expr>-t ékezetteleníti, kis és nagy betűket
                          egyaránt
- %COUNTER              - egy számlálót jelenít meg
- %CNT<expr>            - egy számlálót jelenít meg egyedi hosszal
- %CONCAT<expr1><expr2> - két kifejezést összekonkatenál
- %REPLACE<expr1>       - az <expr1>-beli kifejezésben kicseréli az
          <expr2>         <expr2> összes előfordulását <expr3>-ra
          <expr3>
- %SUBSTR<expr1><expr2> - az <expr1> alstringjével tér vissza <expr2>
                <expr3>   pozíciótól <expr3> hosszon
- %MP3[                 - mp3 file-ok id3v1 tag-jéből kiszedett
        {TITLE}  |        információval tér vissza (ha több információt
        {ARTIST} |        akarunk, akkor többször kell az %MP3[{valami}]
        {ALBUM}  |        kifejezés is): szám, cím, előadó, album neve,
        {YEAR}   |        év, egyebek
        {OTHER}
      ]

Lehetőség van a file létrehozásának dátum részeinek lekérdezésére is:
- %{yyyy}               - négy számjegyes év: 1999
- %yy                   - két számjegyes év: 99
- %m                    - egy/két számjegyes hónap: 1
- %{mm}                 - két számjegyes hónap: 01
- %{mmm}                - hónap neve rövidítve: Jan
- %{mmmm}               - hónap neve teljesen: January
- %d                    - egy/két számjegyes nap: 1
- %{dd}                 - két számkegyes nap: 01
- %{ddd}                - nap neve rövidítve: Sun
- %{dddd}               - nap neve teljesen: Sunday
- %h                    - egy/két számjegyes óra: 0
- %{hh}                 - két számjegyes óra: 00
- %n                    - egy/két számjegyes perc: 0
- %{nn}                 - két számjegyes perc: 00
- %s                    - egy/két számjegyes másodperc: 0
- %{ss}                 - két számjegyes másodperc: 00
```

A számlálók használata kisebb magyarázatra szorul. A %COUNTER és a %CNT között a különbség összesen annyi, hogy a %COUNTER teljes egészében a globális értékekre hagyatkozik az összes előfordulása esetén, így a fix hosszúság tulajdonságban is. A %CNT

ezzel ellentétben a hosszában változó lehet előfordulásonként.
A kifejezések kiértékelése balról jobbra a string végéig történik. Ha egy olyan kifejezést talál, amit nem tud értelmezni, akkor onnantól kezdve borul az egész, és nem fogja tudni továbbértelmezni az egészet. A kifejezések '%'-nal indulnak, ezután jön a neve, ami egyértelműség kedvéért rakhatunk '{' '}' jelek közé (például %yyyy esetén az %yy már értelmes lenne, ezért lesz %{yyyy} jó). A kifejezés végét értelmes kifejezés határozza meg (ezért lehet jó az előbbi példa esetében az %{yyyy}). Ezután jöhetnek a paraméterek, mely sok esetben újabb kifejezések lehetnek, de például %MP3 és %STRING esetében ez egy '{' '}' jelek közé rakott konstans stringnek kell lennie. Ha egy kifejezés paraméterekkel együtt végetésrt, akkor utána rakhatunk tetszőleges stringet, ami meg is fog jelenni, vagy tetszőleges újabb kifejezést. Ha egy paraméternek konstans szám értéket akarunk megadni, akkor az szintén a '%' jel után tehetjuk meg.
Nézzünk egy egyszerű példát:
Eredeti filenév: zene.mp3
Alkalmazott forma: %CNT%2 - %MP3{ARTIST} - %MP3{TITLE} -%TOUP%NAME% EXT
új filenév: 01 - 666 - Paradoxx.mp3 -ZENE.mp3
Ekkor az történt sorban, hogy vettünk egy számlálót (ezt egytől indítottuk) 2-hosszas paraméterrel, majd – ezután az mp3 előadóját szedtük ki, majd '-'-szal elválasztva a szám címét is ezután újabb -, majd a kiterjesztés nélküli név csupa nagybetűkkel, és legvégül a kiterjesztést adtuk hozzá.
Láthatjuk, hogy a formátum meghatározása kissé bonyolult is lehet, de a lehetőségek kevésbé korlátozottak. (Sok commanderben vagy nincs ilyen vagy nem ennyire univerzális.)

2.4. A Beállítóablak

A beállító ablakban van lehetőség a VC konfigurálása, ha el akarjuk kerülni a konfigurációs file szerkesztését. A lehetőségek kb. ugyanazok, de itt talán még több is van (például gyorsbillentyűk beállítása). Az ablak megjelenésével láthatjuk, hogy több fül is megjelenik, ezzel kategorizálva a beállításokat. Nézzük mik ezek a kategóriák: általános beállítások, megjelenéssel kapcsolatos beállítások, gyors könyvtárak, kiterjesztésekhez való program társítás, a csomagkezelők beállításai, betűtípus és színek, végül a gyorsbillentyűk meghatározása. Kezdjük az elején. Zárójelben a konfigurációs file-beli környezeti változót

adom meg.

A következő két fül a konfigurációs file-ban a [general] résznél találhatóak.

Az általános fül alatt találhatunk olyan beállításokat, melyek technikai jellegűek és az általános működéshez kellenek:

- execpath (execpath): ide egy xshellt érdemes megadni olyan paraméterezéssel, hogy utánaírva egy filenevet azt képes legyen elindítani. Ezzel fognak elindulni azok a programok, melyeket a parancssorba írunk vagy amire a panelen entert ütünk, feltéve, ha a parancssor melletti 'X' gomb be van kapcsolva. Alapértelmezett: xterm -e
- packageprefix (packageprefix): ez a program home-jában lebő alkönyvtárat adja meg, ahol a csomagkezelő scriptek találhatóak. Ez azért változtatható, mert a haladó felhasználó trükkösen megadhatja a saját scripjeire mutató könyvtárat, így root jogok nélkül frissítheti a csomagkezelő scripteket. Alapértelmezett : pack/
- tmppath (tmppath): ide egy olyan könyvtárt kell megadni, ahova a felhasználó(k)nak szabad írás joga van. A program futása során ide fog szemetelni. Alapértelmezett: /tmp/
- execcompare (execcompare): ez annak a programnak a neve, melyet akkor hív meg a VC, amikor a felhasználó két file tartalom szerinti összehasonlítását kéri. A programnak úgy kell működnie, hogy a neve után a két megadott file kerül elérési útvonallal. Alapértelmezett: kompare
- extviewer (extviewer): a név magáért beszél. Annak a programnak a neve, melyet alapértelmezésben F3-ra hív meg a program, ha nincs ilyen akkor a belső nézegetőt használja majd. A külső program paraméterben kapja meg a megnézendő file nevét. Alapértelmezett:
- exteditor (exteditor): ugyanaz mint a nézőke, csak itt üresen hagyott mező esetén nem lesz meghívva semmi, és belső szerkesztő sincs. Alapértelmezett: kwrite
- dateformat (dateformat): itt adhatjuk meg, hogy a panel date mezőjében az egyes file-okhoz milyen formaban jelenjen meg a dátum. Ez hasonló mint a Multi Rename Tool-ban:

```
- yyyy              - négy számjegyes év: 1999
- yy                - két számjegyes év: 99
- m                 - egy/két számjegyes hónap: 1
- mm                - két számjegyes hónap: 01
- mmm               - hónap neve rövidítve: Jan
- mmmm              - hónap neve teljesen: January
- d                 - egy/két számjegyes nap: 1
- dd                - két számkegyes nap: 01
```

```
- ddd                - nap neve rövidítve: Sun
- dddd               - nap neve teljesen: Sunday
- h                  - egy/két számjegyes óra: 0
- hh                 - két számjegyes óra: 00
- n                  - egy/két számjegyes perc: 0
- nn                 - két számjegyes perc: 00
- s                  - egy/két számjegyes másodperc: 0
- ss                 - két számjegyes másodperc: 00
```

Alapértelmezett: mm-dd-yyyy hh.nn

- Show Hidden Files (showhiddenfiles): a rejtett fileokat mutassa-e vagy sem (jelenleg mindig mutatja). Rejtet file a '.'-tal kezdődő file. Alapértelmezett: igen
- Ignore Case (ignorecase): ez azt jelenti, hogy maszkolásnál illetve kereséseknél figyelembe vegye-e a kis- és nagybetűk közötti különbségeket. Alapértelmezett: igen
- Human Readable (humanread): ez azt fejezi ki, hogy a size oszlopban a fileok méretét mindig byte-ban, vagy mértékkel kifejezve szeretnénk látni. Alapértelmezett: igen (azaz mértékkel)
- Autocomplete (autocomplete): a parancsok kiegészíthetőségére vonatkozik. Azaz a parancssorban elkezd a felhasználó gépelni valamit, akkor egy már korábban begépelt paranccsal kiegészítse-e. Ugyanez vonatkozik az ActionForm mezőire is (másolás, keresés stb. rákérdezése). Alapértelmezett: nem
- External lister (extls): ez egy trükkös mező, arra szolgál, hogy újra lekérdezze-e a tömörített állomány könyvtár listáját, ha egy mappájába belépünk, vagy ne. Mert ez idő alatt történhettek változások, ezért jó az új lista, de ez természetesen lényegesen lelassítja a belépést. Alapértelmezett: igen (azaz nem csinál új listát mappa váltáskor)
- Bytes of copy (copybytes): az egyszerre másolásra kerülő byte-ok száma. Ez változtatható, hogy a felhasználó optimális sebességet érhessen el másolás közben. Alapértelmezett: 32765
- Number of threads (maxthreads): egyenlőre még nincsenek thread-ek (szálak), de később ennyiben lesz maximalizálva az egyszerre indítható szálak (másolás, törlés stb.), jelenleg ez szabályozza, hogy mennyi Viewer lehet egyszerre megnyitva. Alapértelmezett: 5

Az általános beállításokat követi a Visual fül, ahol a megjelenítéssel kapcsolatban tudjuk testreszabni a Commanderünket:

- colwidthX (colwidthX): termeszetesen az oszlopok szélességét lehet beállítani vele. X az egy 1 és 5 közötti természetes szám. Természetesen a panel feletti csúszkákkal is

szabályozhatjuk, de itt el is menthetjük, mivel ez a tulajdonság kilépéskor nem mentődik el automatikusan. Alapértelmezett: colwidth1=150, colwidth2=75, colwidth3=85, colwidth5=64, colwidth4=230

- colwhatX (colwhatX): az egyes oszlopok milyen tulajdonságát jelenítsék meg a file-oknak. Lehet akár ugyanaz is. Alapértelmezett: colwhat1=name, colwhat2=size, colwhat3=rights, colwhat5=date, colwhat4=link
- Height of fonts (fontsize): a panel és körülötte levő szövegek betűinek mérete (magassága). Van külön font állító, de az egyenlőre nem menthető el. Alapértelmezett: 12
- Weight of fonts (fontweight): a panel és környezetének szövegeinek betűinek súlya (vastagsága). Ez a jobban olvasatóság miatt kell. Alapértelmezett: 75 (vastag)
- Height of viewer's font (viewerfontheight): a nézőkének a betűmérete. Alapértelmezett: 12
- Distance between elements (dist): Ebben határozhatjuk meg a különböző elemek távolságát, azaz hogy mekkora térköz lehet közöttük. Alapértelmezett: 2
- topshift (topshift): a panel egyes sorainak pozíciója a cellákhoz képest. Ez azért szükséges, mert néhány rendszeren elcsúsztak a szövegek. Alapértelmezett: 2
- iconset (iconset): kiválaszthatjuk a file-okhoz rendelt ikonok stílusát: default, KDE, Gnome. Alapértelmezett: default
- Directory names like this (directory_mode): a listában a könyvtárak nevének jobban elkülönülésére szolgál, megadhatjuk a könyvtárnevének megjelenési környezetét. Ebben a stringben a %DIR% határozza meg a nevet, körülötte teszőleges string lehet. Alapértelmezett: %DIR%, konfigurációs file-beli alapértelmezett: [%DIR%] (azaz a könyvtárnév '[' és ']' jelek közé kerül)
- Directory String (dirstr): Ez azt mondja meg, hogy a fileméret oszlopba a méret helyett mit irjon könyvtárak esetén. Alapértelmezett: Dir, konfigurációs file-beli alapértelmezett: <DIR>

Ezután jön egy bonyolultabb téma, mégpedig a kiterjesztésekhez filerendelés. Ez a rész a Extensions, a konfigurációs file-ban a [extensions] résznél található.

Ha a panelen egy adott kiterjesztésű filera duplaklikkelünk vagy entert ütünk, és az nem tömörített file, akkor ezt a programot indítja el paraméterül átadva azt a file-t, amin entert ütöttünk.

Ennek a résznek a felépítése a következő: két részből áll egy hozzárendelés, egy

kiterjesztésből és egy kezelőből. Legyen a hozzárendelés neve avi, ekkor a kiterjesztés rész az alábbi: avi_ext=.avi, a kezelő rész: avi_handle=xterm -e mplayer. Ezzel megadtam, hogy az .avi kiterjesztésű file-okhoz az mplayert indítsa.

Hasonló a következő rész is, de ez már kifejezetten a tömörített file-ok kezelésére vonatkozik, így különleges elbánásban részesülnek, különleges kezelőprogramok révén. Ez a rész a Package Extensions, a konfigurációs file-ban a [package] résznél található.
Az itt szereplő hozzárendelések is rendelkeznek névvel, ami a tulajdonságok előtt találhatóak meg, és az első megjelenéskor automatikusan létrejönnek. Ezek alapján egy sor felépítése az alábbi: <név>_<tulajdonság>=<érték>. A nevet mi választhatjuk az tulajdonságok és értékeik az alábbiak lehetnek:

- ext: a hozzárendelés alapja, egy kiterjesztés (string)
- handle: a kezelő program/script neve. Ez található meg a pack/ (packageprefix) könyvtárban (string)
- copyin: a használt tömörítő program lehetőséget ad-e a tömörített állományba való másolásra (azaz tömöríteni) (bool)
- copyout: a használt tömörítő program lehetőséget ad-e a tömörített állományból való kimásolásra (azaz kicsomagolni) (bool)
- rm: törlési lehetőség (bool)
- dir: az adott program milyen file listát szolgáltat. Ha a filelista magában foglalja a könyvtárakat is és azok összes elérési útját, akkor lehet true, különben false. Ha nem vagyunk biztosak a dolgunkban, akkor legyen false, mert ez általánosabban lehet jó. Ace, bz2, gz, rar, tgz lehet true tesztelés alapján, de zip, deb, rpm szigorúan false lehet. (bool)
- run: van e lehetőség file-t futattni az állományon belül. Ez csak deb, rpm és hasonló csomagoknál lehet hasznos, ahol így a csomagba lépve lehetősg nyílik telepítésre/frissítésre.

Nézzünk egy példát, ahol .ace-hez akarunk egy kezelőt csinaálni. A kezelő neve legyen ace, a kiterjesztés .ace, kezelőprogram uace (a pack/-ban, ez egy stript), a script csak listazni tud (alapkövetelmény), kimásolni és a filelista tartalmazza a fileok elérési útját:

```
ace_ext=.ace
ace_handle=uace
```

ace_copyin=false
ace_copyout=true
ace_rm=false
ace_dir=true
ace_run=false

Fontos megjegyezni, ha valamelyik tulajdonság hiányzik a konfigurációs file-ból akkor az vagy false lesz bool esetén, vagy üres string esetén.
Ejtsünk néhány szót a kezelő programok működéséről. Ezeknek a paraméterezésére elég szigorú szabályok vonatkoznak. Első paraméternek mindig a végzendő műveletet kapják meg: list, copyin, copyout, rm, run. Ezután mindig szükséges megadni a tömörített állomány nevét amin dolgozunk. Ezután a paraméterek száma változó lehet értelem szerűen. Például copyoutnál, szükséges a tömörített állomány neve (magában foglalva az elérési utat természetesen), majd hogy hova csomagoljuk azt ki (szintén útvonal+név). Azaz például az ace esetén: uace copyout hello.ace hello.txt /home/btuska/hello_new.txt.

A most következő fül szintén a kinézettel kapcsolatos, de már nem fért a Visual fülre. Ez a menüpont a Fonts & Colors, mivel csak a színek kerülnek elmentésre, a fontok nem, ezért ezt a részt a konfigurációs fileban a [color] rész alatt találjuk meg. Itt értelemszerűen az egyes komponensek színét illetve betűtípusát tudjuk meghatározni. Van két alapértelmezett érték, az egyik sötét a másik világos.

Most következik a gyorsbillentyűk beállítására szolgáló fül, a ShortCuts. Ez a rész nincs benne a konfigurációs file-ban, de természetesen elmentésre kerül, mégpedig a shortcuts file-ban. Itt lehetőség van a legtöbb funkció gyorsbillentyűjét megváltoztatni, azokét melyeket menüből el lehet érni.
Láthatunk egy listát, melynek elemein duplaklikkelve, vagy entert ütve, vagy rápozicionálva és a change gombra kattintva feldobódik egy új ablak melyben az új kombinációt leütve érvényesíthetjük azt. Ha ebben a dialógus ablakban ESC-et ütünk, akkor nem történik változás. Meg is semmisíthetjük a beállított gyorsbillentyű beállításunkat, úgy ha a törlendő elemre ráállunk, majd delete gombot megnyomjuk.

A változok beállítása némely esetben azonnal érvényesítődik, de sokuk nem, csak akkor

amikor a „Save and Accept” gombra kattintanak. A Close gombra azoknak a változóknak az érvényessége nem szűnik meg, melyek azonnali hatást eredményeztek, a többi érvénytelen marad. Ha elmentettük, akkor a régi konfigurációs file elmentődik (de csak egy példányig visszamenőleg), majd az új létrejöttével, azt újrabeolvassa, így a változók új értéket kapnak.

2.5. A file műveletek

A file műveletek közé lehet azokat a műveleteket sorolni, melyek valamilyen hatással vannak a panelre és az azokban elhelyezkedő file-okra. Például másolás, mozgatás, átnevezés, törlés, keresés, könyvtárméret, file/könyvtár létrehozása, jog- és felhasználó megváltoztatása, szimbolikus linkek kezelése. Ezek közös jellemzője (könyvtárméret kivételével), hogy a műveleti tulajdonságok beállítására egy ActionFormot használnak. Ez egységes minden művelet esetén, de a nem értelmes illetve nem alkalmazható tulajdonságok nem lesznek aktívak. Ezeket a tulajdonságokat az egyes műveleteknél külön elmondom.
Alapvetően van két inputablak, és három pipálási lehetőség (szimbolikus linkek követése, rekurzivitás, attribútumok megőrzése)
Egyik művelet előtt sem nézi meg az összméretét azoknak a file-oknak, amin a műveletet végezni fogja, mert ez némely esetben túl lassú lenne. Így a művelet teljes hosszára nem tud becslést mondani, csak egyes részeredményekre (kijelölésekből mennyit hajtott végre, egy alkönyvtár file-jain hol tart, egy file-ban hol tart -> 3 progressbar lesz). Az aktuális műveletet egy ProgressForm-on nézhetjük végig. Itt lehetőség is nyílik a művelet megszakítására, mely felhasználói megerősítést kér, és csak ezután szakítja meg vagy folytatja a műveletet.

2.5.1. Másolás

Ez a művelet olyan, hogy mindkét panelra kihatása lehet. Itt megkülönböztetünk forrás- és cél panelt. A forrás ahonnan másolunk, a cél pedig ahova. Két másolás lehetséges, az egyik a sima másolás, tehát egyik panelról a másikra, a másik másolás pedig a helyben másolás, ekkor megegyezik a forrás és a cél panel. Az általános másoláshoz alapértelmezésben az F5-öt, míg a helyben másoláshoz a SHIFT+F5-öt használhatjuk.
A másolás forrása lehet normál filelista, tömörített állomány (megfelelő beállítás esetén), keresési eredmény illetve egyéb VFS (ftp). A cél szintén lehet normál filelista, tömörített

állomány, vagy VFS. A másolás alatt kijelzi a másolás sebességét (kbyte/sec), és a másolás alatt levő file-hoz megbecsüli a hátralevő időt (sec).
Ennél a műveletnél az ActionForm a következő paraméterekkel rendelkezik:

- a célt meghatározó inputsor: ez sima másolás esetén alapértelmezésben a cél panel aktuális mappája, helyben másolás esetén a forrásé. Ez tetszőlegesen megváltoztatható, kivéve, ha a cél ftp vagy tömörített állomány.
- A cél maszkját meghatározó inputsor: ez egy egyszerűsített maszk lehet (bővebben a maszk alfejezetben), másolás esetben ez egy „*”, hogy mindig az az aktuálisan másoladó file legyen. Helyben másoláskor ez annak a filenak a neve kerül ide amin éppen állunk (egy kijelölés esetén, több esetén ez is „*”).
- Follow symlinks: a szimbolikus linkek követése, ez szimbolikus link másolása esetén dönti el, hogy linket csináljon vagy az másolja ahova mutat. Könyvtár link esetén, hogy belemenjen-e abba a könyvtárba ahova a link mutat.
- Preserve attributes: ez azt jelenti, hogy másolás során a file-ok attribútumai meg maradjanak-e, vagy a rendszerre jellemző default értékekkel jöjjenek létre.

A másolás egy biztonságos művelet, tehát törlés nem történik. Arra vigyázzunk, hogy ahonnan másolunk az nem lehet prefixe annak ahova másolunk, ha ilyet tennénk a program nem engedi meg ezt. Néha ez zavaró lehet, de sok kellemetlenségtől megkíméli az embert. Ha törlés nem is lehet, de adatvesztés igen, ha olyan helyre másolunk, ahol az adott file már létezik. Ekkor a program természetesen figyelmeztet, hogy a cél már létezik, ilyen esetben több lehetőségünk is van: felülírás, másolásból kilépés, nem felülírás, hozzáírás. Ezenfelül, ha több file-t is másolunk és több cél is létezik, akkor zavaró lehet, hogy a program mindig megkérdezi, hogy mi legyen, ez esetben a felülírást és a hozzáírást érvényesíthetjük az összes esetre.

Általában a műveletek közben előfordulhat, hogy valami miatt az, vagy annak egy része nem sikerült, ekkor a program ezt egy hibaüzenettel jelzi, legtöbbször valamilyen jogi probléma miatt. Ekkor ez vagy figyelmen kívül hagyjuk, vagy kilépünk a műveletből, végül figyelmen kívül hagyjuk és a további üzeneteket is.

2.5.2. Mozgatás/Átnevezés

Ez a két művelet teljesen hasonló, mint a másolás illetve helyben másolás, azzal a különbséggel, hogy a forrás megfelelő jogok esetében törlődni fog. A forrás és vél itt már nem lehet semmilyen VFS (ftp, vagy tömörített file). A mozgatás alapértelmezésben F6-tal, az átnevezés alapértelmezésben SHIFT+F6-tal történik.
ActionForm-beli különbség, hogy itt mozgatás esetén a szimbolikus linkek követése és a konzisztencia megőrzése érdekében, csak a szimbolikus linkek kerülnek mozgatásra, az hogy hova mutatnak lényegtelen. Az attribútum megőrző hatásnak elsőre nincs sok értelme, de lehetséges, hogy különböző partíciók között mozgatva a mozgatás művelet felbomlik másolásra és egy ezutáni törlésre. Ekkor a másolsához kell attribútum. Ugyanez az eset, mikor a forrás helyen nincs törlés jogunk, ekkor mozgatás helyett egyszerű másolás hajtódik végre.

2.5.3. Törlés

A törlés csak abban az esetben van hatással mindkét panelra, ha ugyanazt a listát mutatják. A törlés ActionFormjához nem tartozik sem input ablak, sem pedig egyéb tulajdonság. Szimbolikus link esetén csak a link törlődik, gyakran ahova mutat, azt nincs is jogunk törölni.
A törlés ugyanúhy mint a mozgatás és másolás mindig vagy az aktuális file-ra hat, ha nincs kijelölés, ellenkező esetben az összes kiválasztott file-on végrehajtódik.
A törlés alapértelmezésben F8-cal történik.

A mozgatásnál és törlésnél kifejezetten ügyeljük arra, hogy mit teszünk! A rendszer károsodádához is vezethet, ha nem megfelelő file-okat törlük illetve mozgatunk. Különösen veszélyes ha root jogokkal futattjuk a programot (ez általában amúgy sem ajánlott)!

2.5.4. File/könyvtár létrehozása

Hogy nem csak romboljunk hanem építsünk is, lássuk mi kell egy file illetve könyvtár létrhozásához. File-t SHIFT+F4-gyel, könyvtárat F7-tel tudunk létrehozni. Az ActionForm-nak egyetlen bemeneti mezője van, mindenféle tulajdonság nélkül, itt adhatjuk, meg a nevét

a létrehozandó objektumnak. A létrehozott file/könyvtár a rendszerre jellemző jogokkal, hogy létrejönni. A panelben levő kiválasztásoknak itt nincs értelme, és a művelet végére megszűnnek. A művelet eredményeképpen az aktuális panelban létrejön egy üres file/könyvtár.

2.5.5. Filekeresés

A file keresés, mint a neve is mutatja file-ok keresésére használható, könyvtárakat a keresés eredménye nem tartalmaz. Ahhoz, hogy a file kereső ablak előjöjjön alapértelmezésben az ALT+F7 billentyűket kell leütni, természetesen menüből is elérhető. A kereső ablak két inputsort tartalmaz, valamint egy rekurzív keresésre lehetőséget adó checkboxot. A filekeresés csak normál panellista esetén működik.
Figyelem: az alkönyvtárakban való kereséshez, kapcsoljuk be a rekurzív módot!
File-okat névminta illetve tartalom alapján kereshetünk. A névminta egy maszk, ezt az első inputsorban adhatjuk meg (lásd a maszkokról szóló fejezetet). A második inputsorban adhatjuk meg azt, hogy a file-nak a tartalmában milyen stringnek kell előfordulnia. Ez is egy maszk lehet. Ha nem adunk meg semmit, akkor nem történik tartalom alapján szűrés. (Egyenlőre több szempont alapján nem nincs lehetőség szűrni, azaz fileméret, dátum stb.)
A keresés mindig abban a könyvtárban értendő, amiben éppen az aktuális panel van. A kijelölések nem lesznek figyelembe véve. A keresés eredmény az aktuális panelban jelenik meg, úgy hogy a filenevek kiegészülnek az útvonalukkal is. Ezekután gyakorlatilag minden olyan művelet megengedett nála, ami sima panel esetén is (az észszerűség határain belül).

2.5.6. A könyvtárméret megállapítása

Ha nem csak a file-ok méretét szeretnénk tudni, akkor lehetőség van a könyvtárak méretének megállapításához. Ehhez (megváltoztathatatlanul) a SPACE billentyűt kell megnyomnunk, vagy a file-on a nézőkét előhozó gombot, de ezt a könyvtáron csinálva. Ekkor a kurzor alatti könyvtár mérete az eddig '<DIR>' felírat helyett jelenik meg a size oszlopban.
Lehetőség van az aktuális könyvtár összes alkönyvtárának méreteinek megjelenítésére, ez alapértelmezésben az ALT+SHIFT+ENTER billentyűkombinációval történik.
A könyvtár méretének kiszámításához a szimbolikus linkeket nem követi, mert az esetek legnagyobb részében végtelen ciklusra futna.

2.5.7. A jogok beállításai

A jogok, a szokványos jogok beállítása, külön könyvtárra és külön file-okra. Alapértelmezett billentyű nincsen rá, de menüből elérhető. Két inputsora van, plusz a szimbolikus követés és rekurzió lehetősége. Az első inputsor a file-okra a második a könyvtárakra vonatkozik. Erre azért van szükség, mert legtöbb esetben a file-okra nem kell futtatási jog, míg a könyvtáraknál igen. A jogok beállítása csak normál panellista esetén működik.A megadás módja a következő:

ugsrwxrwxrwx

Sorrendben: setuid bit (u), setgid bit (g), sticky bit (s), owner olvasás (r), owner írás (w), owner végrehajtás (x), csoport olvasás (r), csoport írás (w), csoport végrehajtás (x), egyéb olvasás (r), egyéb írás (w), egyéb végrehajtás (x). A végrehajtás során a kijelölt file-okon megy végig, könyvtár és rekurzió esetén először magára a könyvtárra érvényesíti a jogokat, majd a tartalmára is (ez a sorrend azért van, mert többségében a felhasználó nem magától veszi el a jogot (ez esetben nem tudna érvényt szerezni a tartalomra), hanem inkább plusz jogokat ad, és ez így jobban lehetséges).

2.5.8. A felhasználó megváltoztatása

Ez egy igen ritkán használt képesség, mert root jogok kellenek a file-ok tulajdonosának megváltoztatásához. Az ActionForm-nak itt is ugyanazok a mezői vannak, mint a jogok esetében. De itt már csak egy érték lesz érvényes, azaz nincs külön lehetőség file-ok és könyvtárak tulajdonosait külön megváltoztatni. A megadás módja a következő: uid:gid. Uid/gid azt jelenti, hogy ide csak szám érték kerülhet kettősponttal elválasztva. (Később lehet, hogy lesz név alapján is. Puskázni az /etc/passwd és /etc/group file-okból lehet.) A file tulajdonosának csak normál panellista esetén működik.

2.5.9. Szimbolikus linkek kezelése

Létrehozás:

Ebben az esetben a kurzor alatt levő file-ról csinál egy szimbolikus linket alapértelmezetten a másik mappába ugyanolyan néven. Ez a két inputsorban adható meg. Az első az mondja

meg, hogy a link hova mutasson, alapértelmezésben a kurzor alatti file neve útvonallal együtt (azaz ez az eredeti file). A második inputsor a link neve útvonallal (azaz ez az újonnan megjelenő file).

Szerkesztés:
Egy már létező linken tudjuk ezt végrehajtani. Az ActionForm egyetlen inputsora az éppen a kurzor alatt levő link célja (linktarget).
A szimbolikus linkek kezelése csak normál panellista esetén működik.

2.5.10. Egyéb panelfunkciók

Panel tartalmának frissítése:
Lehetőség van a panel újraolvasására is. Ez hasznos lehet, ha változások történtek a file strukturában, miközben használtuk a programot, de nem váltottunk könyvtárat, vagy ha mást csináltunk, és közben a program nem volt aktív.

Panel tartalma a másik panelbe is:
Alapértelmezésben a CTRL+T-vel érhetjük el. Arra jó, hogy az éppen nem aktuális panelban is ugyanaz a tartalom jelenjen meg mint az aktívban. Ez akkor jó, amikor a két panel tartalma távol van egymástól, de közel akarjuk hozni őket egymáshoz valamely műveletvégzés szempontjából.

File-ok tartalom szerinti összehasonlítása:
A két panel éppen aktuális file-jaival hívja meg az összehasonlító programot. Persze itt megadható más program is, ami esetleg nem összehasonlításra jó, de alapértelmezésben a „kompare” grafikus tartalomösszehasonlító programot hívja meg ehhez a művelethez.

2.6. Egyebek

2.6.1. A maszk és az egyszerűsített maszk

A maszk egy string-formátum, amit akkor kell használni, amikor nem csak egy file-on végzünk műveletet, hanem többön is. Ekkor meg tudunk adni az adott file-ra jellemző

dolgokat. Ez a maszk esetén bonyolultabb, míg az egyszerűsített maszk esetén különféle korlátozások vannak. Általában külön esetekre használhatóak.

A maszk

A maszk esetén a feladatot úgy lehetne meghatározni, hogy sok-sok stringből a maszk alapján annak egy részhalmazát választjuk ki. Ez történik a kiválasztás maszk esetén hatására (lásd ide kapcsolódó fejezetet). A másik eset az előzőnek egy részfeladata, az hogy egyetlen stringre meghatározni az illeszkedést. Ez történik file-keresés esetében.
Nézzük, hogyan kell kinéznie egy maszknak, mi benne a speciális, milyen szabályokat kell betartani. A maszk egy olyan string melyben speciális helyettesítő karakterek lehetnek. Ezek a karakterek a következők: '*' - bármely karakter helyettesítése bármilyen hosszon, '?' egy tetszőleges karakter. Illeszkedés akkor történik amikor a helyettesítő karaktereknek létezik olyan helyettesítése melyre a maszk meg fog egyezni a vizsgálandó stringgel. Az illeszkedés vizsgálata természetesen az egész stringre vonatkozik, nem pedig annak egy részére. Természetesen az a string melyre az illeszkedést vizsgáljuk, nem tanácsos, hogy '*'-ot tartalmazzon, persze lehet, de ekkor a maszkban levő csillag attól még az általános helyettesítést fogja jelenteni. Ebben az esetben több illeszkedés is lehet, mint amennyit mi eredetileg is szeretnénk. Például:

string: Az ég tele van *-okkal!

maszk: *an *-okk*

Ekkor van illeszkedés, de ha a string: „Az ég tele van csillag-okkal!” is jó lesz, pedig adott esetben ezt nem szeretnénk.
Megjegyzés: a file-tartalom keresés esetében nem kell azt a szabályt betartani, hogy '*'-gal kell kezdődnie és végződnie, ahhoz, hogy a string belsejében is megtaláljuk az illeszkedést, mert ezt a program kényelmi szempontokból automatikusan megteszi.

Az egyszerűsített maszk

Az egyszerűsített maszk lényegében és funkciójában is eltér a sima maszktól. A fentebb tárgyalt sima maszk mintaillesztésre jó, míg az egyszerűsített változata erre nem, hanem egy stringből csinál egy másikat. Itt is két főszereplője van az eseménynek, az egyik maga a string, a másik pedig a maszk maga. Ez a maszk már csak egy speciális jelet tartalmazhat,

ami a '*'. Ennek az a jelentése, hogy az egész stringet behelyettesíti a visszaadandó stringbe. Ez akkor hasznos, mikor másolunk több file-t, és az szeretnénk, hogy valami alapján ezek elkülönüljenek, ezt alapból meg is teszik a saját nevük alapján, azaz ennek az egyszerű '*' fog megfelelni. De emelett mész szükség lehet némi átnevezésre is.

2.6.2. A kiválasztás, kijelölés

A kiválasztás akkor szükséges, amikor azt szeretnénk meghatározni, hogy egy műveletet mely file-okra hajtsunk végre. Ha nem jelölünk ki semmit, akkor a kurzor alatti file-on tudunk csak műveletet végezni. A kijelölést több úton végezhetjük el, különböző szabályok szerint. A legkézenfekvőbb módja, hogy egyesével jelöljük ki a file-okat. Ezt az INSERT gombbal tehetjük meg (ez nem változtatható). Ez néha túl macerás, ezért lehetőség van csoportos kijelölésre is. Alapértelmezésben a CTRL+A-val tudjuk az összes file-t kijelölni. Ha bonyolultabb szabály szerint szeretnénk kijelölni, akkor használjuk az alapértelmezésben '+'-ra előjövő maszk általi kijelölés megfordítást (a maszkról az előző fejezetben olvashatunk részletesebben). A kijelölés megfordítás alapértelmezésben a '*'-gal történik (jó, ha nincs kijelölve semmi, akkor mindent kijelöl, még a könyvtárakat is). Ha tetszőleges kijelölést szeretnénk megszüntetni, akkor azt a '-'-szal tehetjük meg.

2.6.3. A rendezésről

A file-ok a panelokban alapértelmezésben név szerint rendezve jelennek meg. A könyvtárak és a file-ok elkülönülve vannak, és az összes rendezésre az a jellemző, hogy a könyvtárakat és a file-okat külön rendezi, és a könyvtárak mindig legelöl vannak. De a lista legelső eleme az mindig a '..'.
A rendezés szabályát meg lehet változtatni, illetve a sorrendet megfordítani. Erre lehetőség nyílik a File menüből (ezért gyorsbillentyűvel is akár) és a panelek feletti kis elemek (HeaderControl) segítségével. A fejléc egy oszlopára klikkelve meghatározhatjuk az azon oszlop szerinti növekvő rendezettséget, második klikkre a sorrend irányát fordíthatjuk meg. Nem minden oszlop szerint lehet rendezni, csak név, méret és dátum szerint.

2.6.4. „ALT”-os keresés

Ha egy olyan helyen (könyvtárban) tartózkodunk, ahol sok file van, és mi egy bizonyos file-t keresünk, mert azzal akarunk műveletet végezni, akkor nem érdemes a keresést használni (ALT+F7), hanem az „ALT”-os kereséssel egyből oda juthatunk. Ezt úgy tehetjük meg, hogy az ALT billentyű nyomvatartása mellett elkezdjük begépelni a keresendő file nevét. Amíg illeszkedést talál a program, addig mindig az első illeszkedésre ugrik. Ezt a kurzor poziciója+1 sortól nézi, ha nincs illeszkedés, akkor a lista elejétől kezdi a kurzor pozícióig.

2.7. A konfigurációs file-ok

A konfigurációs file-ok azok a file-ok, melyekben a felhasználók saját beállításaik találhatók. Kétféle file van. Az első a fő konfigurációs file, ez alapértelmezésben érkezik a rendszerrel, mindig megtalálható a bináris mellett. A többi konfigurációs file csak akkor jön létre, amikor a felhasználó elkezdi használni a programot. A fő beállítások file-ja, azért fontos, mert a program ez alapján határozza meg azokat a parméterértékeket, melyeket el kell menteni. Azok az értékek, melyek nem szerepelnek a file-ban, nem kerülnek el mentésre. Nézzük a konfigurációs file-ok kezelése, hogyan is zajlik.
A felhasználó programindításakor a program elkezdi keresni a fő beállítás file-ját, először a felhasználó home könyvtárának ($HOME) a „.vc” alkönyvtárában (a további file-ok is ide kerünek). Ha itt van a file, akkor azt beolvassa, majd a többit is. Ha nincs itt a file, akkor abban a könyvtárban keresi, ahol a program binárisa található, ha itt találja, akkor bemásolja az eredeti keresés helyére, és onnan olvassa be. Ha nem találja meg a bináris mellett, akkor hibaüzenet konstatálása következtében az alapbeállításokkal indul el. (Ebben az esetben nem lesz lehetőség a felhasználó által megváltoztatott értékek elmentésére.) Most vegyük sorrendbe a konfigurációs file-okat név szerint megmutatva, hogy melyik mire jó, és mikor kerül elmentésre. (Ezen file-ok helye: $HOME/.vc/)

A „bookmark” nevű konfigurációs file

Ebben a file-ban a felhasználó gyorskönyvtárai vannak elmentve. Tetszőleges sort tartalmazhat, minden egyes sor egy-egy könyvtár neve. Betöltése a fő konfigurációs file beolvasása után történik meg a program indulásakor, illetve amikor a felhasználó

újraolvasatja a file-t. Elmentése a programból való kilépéskor, illetve új bejegyzés létrehozásakor automatikusan megtörténik. Mivel a bookmark bejegyzések szerkesztésére külön felületen egyenlőre nincs lehetőség, ezért a hozzáadás kivételével a file szerkesztésével tudjuk a bejegyzéseket módosítani illetve törölni.

A „történet” file-ok (doon, doon2, history)

Itt olyan dolgok kerülnek elmetésre, melyeket a felhasználó az egyes inputsorokba írt be. Ez nevesítve a parancssor, a műveletek esetében felbukkanó ActionForm két inputsora (több nincs, de az ActionForm elég sok helyen szerepel). A history file a parancssorhoz tartozik. Az azonos sorok csak egyszer kerülnek elmentésre, és maximálisan 20 különböző sor lehet, az újak kiszorítják a régebbi sorokat (egy majdnem FIFO módszert használva). Az ActionForm két inputsorához tartózó file-ok a doon és doon2 (az első illetve a második sorhoz). Ezek hasonlóan működnek, mint a parancssor, de itt maximum csak 15 elem lehetséges (ennek oka a sebesség miatt van, lásd a fejlesztői dokumentációban).

A positions file

Ez egy érdekes eset, mert már a program sokkal régebbi verzióiban is megjelent, de különösebb funkcióval nem bírt. Ez a file nem csinál mást, mint hogy kilépéskor elmenti az ablak pozicióját és méretét, melyet a következő indítás során alkalmazni fog. A file hiányában a program a desktop közepére helyezi el magát (bár ez ablakkezelőtől függhet), az alapértelmezett méretekkel (ez jelen esetben 717x504).

A shortcuts file

Mint a neve is mutatja, ebben a file-ban találhatóak meg a felhasználó által beállított egyes menük gyorsbillentyűi. Ez minden kilépéskor illetve a konfigurációs ablak mentése esetén elmentődik. Tartalmazza az összes a konfigurációs ablakban beállítható gyorsbillentyűk kódjait. Ezeket mindenféleképpen csak a programon keresztül érdemes megváltoztatni, a file egyéb szerkesztése nem javasolt, sor törlés esetén az egész gyorsbillentyű-kezelő rendszere a felhasználónak összekavarodhat, egyéb szerkesztés is nem várt hatást eredményezhet!

A vc.cfg a fő konfigurációs file

Ez a file kitüntetett szereppel bír a többihez képest. Ez a file mindig létezik csak a helye lehet eltérő. Ha ez a file nincs, akkor az eddigi konfigurációs file-beli funkciókon kívül semmilyen más adat elmentésére nem leszünk képesek a programon keresztül, és a programot mindig csak az alapbeállításokkal tudjuk majd használni. Ez a file tartalmaz minden egyéb más eddig nem tárgyalt beállítást, amit a konfigurációs ablakban be tudunk állítani. Ez magábafoglalja az általános információkat, a vizuális megjelenítéssel kapcsolatos dolgokat, a kiterjesztés- és csomaginformációkat.

A fő konfigurációs file felépítése

A file négy nagyobb részből áll (ezek '[' és ']' jelek között találhatóak), mindegyik nagy rész saját alváltozókból áll, melyeknek értéket tudunk adni. A váltózók és a nagy részek nevei között kommenteket helyezhetünk el, melyek '#'-sel kezdődnek (lehet '/' is, mindkettő csak sor kezdetén), a file egy sora így vagy egy nagyrész nevét, vagy egy váltózót az értékével, vagy '#'-kal kezdődő kommentet tartalmaz. Az alapértelmezett konfigurációs file-ban már eléggé sok komment el van helyezve, minden változó illetve egy-egy adott nagyrész magyarázó szövegét tartalmazza, ezzel segítve a „linuxosabb beállítottságú” felhasználókat, hogy ne a programon keresztül, hanem közvetlenül a konfigurációs file-ban állítgasság a változókat. Az új konfigurációs file érvényesítését az Options menüpontból is el tudjuk végezni, nem szükséges a program újraindítása.

A nagyrészek

Ezek az alábbiak [general], [color], [package], [extensions].
A [general] a legnagyobb, legtöbb változót magábanfoglaló rész. Ennek a váltózói:

- execpath: ez a név megtévesztő lehet, mert nem egy útvonal, hanem egy program, amit a VC akkor hív meg, amikor a felhasználó az 'X' gomb (a parancssor mellett) benyomása mellett egy programot indít el. Ekkor ezzel a programmal hívja meg az indítandó programot. Tehát az execpath egy x-shellnek kell lennie. Ez akkor szükséges, amikor karakteres programot szeretnénk futtatni. A VC nem tudja egy futtatandó programról eldönteni, hogy az grafikus vagy konzolos, ezért a felhasználóra hárul a feladat, hogy ezt

eldöntse, és kiválassza a megfelelő indítási környezetet. 'X'-es indítása esetén az alábbi rendszerhívás történik meg: execpath %FILENAME%. Alapértelmezett értéke: xterm -e. E helyett választhatunk más x-shellt is, például: konsole -e (ez lassabbm, de több funkcióval bír).

- colwidthx: A panelek öt oszlopa közül az x-ediknek állíthatjuk be a szélességét. Ebből is látszik, hogy x 1 és 5 közötti természetes szám lehet. Alapértelmezésben ez a szélesség 64. De az alapértelmezett konfigurációs file-ban ezek az alábbiak az alapértékek: colwidth1=150, colwidth2=75, colwidth3=85, colwidth5=64, colwidth4=230.
- colwhatx: A panelek öt oszlopa közül az x-ediknek állíthatjuk be a tartalmát. Az x itt is 1 és 5 közötti természetes szám lehet. (Az alapértelmezett értékeknek nincs értelme :) A konfigurációs file alapértelmezett értékei: colwhat1=name, colwhat2=size, colwhat3=rights, colwhat5=date, colwhat4=link. Az értelmes értékek is ezek lehetnek, azaz: name (név), size (méret), rights (jogok), date (dátum), link (linktarget, azaz ha a file link, akkor az ahova mutat, tömörített file-ok esetén a tömörített file-ban file elérési útja (technikai okok, lásd fejlesztői dokumentáció)).
- packageprefix: a tömörített file-ok kezeléséhez szükséges script file-ok relatív elérési útja a bináris könyvtárából számítva. (A script file-ok felépítéséről lásd a fejlesztői dokumentációt.) Az alábbi hívás történik a tömörített file kezelésekor: „program_path+packageprefix+package_handler_name opciók %FILENAME% egyéb_opciók”. Alapértelmezett érték: pack/.
- tmppath: Ez a file egy olyan könyvtár elérési útja, melybe a felhasználónak írási joga van. Ide fog a program „szemetelni”. Azaz ennek az alábbi könyvtárába: fctmp{uid}. Alapértelmezett értéke: /tmp/.
- extviewer: Ha a beépített nézőke nem felel meg (F3), akkor itt külső programot rendelhetünk a nézegetéshez. Ha a mezőt üresen hagyjuk, akkor F3-ra is a belső nézőke jeleníti meg a file tartalmát. (SHIFT+F3-ra mindig a belső nézőke jelenik meg.). Hívás az alábbi módon: extviewer %FILENAME%. Az alapértelmezett értéke:
- exteditor: hasonló mint az extviewer, de itt nincs beépített szerkesztő, azaz ha F4-gyel szerkeszteni kyvánunk egy file-t, akkor mindenféleképpen adjunk meg egy szerkesztőt. Hívás: exteditor %FILENAME%. Alapértelmezett érték: . A konfigurációs file-beli alapértelmezett értéke: kwrite. (Ez egy intelligens program, kicsi (lassú), highlightol különböző típusú szöveges file-okat (c, html stb.).
- showhiddenfiles: a rejtett file-ok ('.'-tal kezdődőek) mutatásának a lehetősége lenne, mert

jelenleg hatástalan. Ez az érték bool érték (igaz, vagy hamis). Az alapértelmezett értéke: false. A konfigurációs file-beli alapértelmezett értéke és jelenleg mindig: true.

- hotdirx: ez egy inaktív változó halmaz, csak úgy bennemaradt (egykori bookmark értékek). TÖRLENDŐ.
- execcompare: ez a program fogja elvégezni az összehasonlítást két file között tartalom szerint. A program hívása az alábbi formában történik: execcompare %FILENAME1% % FILENAME2%. Alapértelmezett értéke: kompare. Remek grafikus file-tartalom összehasonlító program, de KDE-s módra tetűlassan indul.
- dateformat: A panelek date oszlopában megjelenő file-ok dátumainak a kiírási formátuma. Az alábbi beépített dátum-elemeket használhatjuk:

```
- yyyy              - négy számjegyes év: 1999
- yy                - két számjegyes év: 99
- m                 - egy/két számjegyes hónap: 1
- mm                - két számjegyes hónap: 01
- mmm               - hónap neve rövidítve: Jan
- mmmm              - hónap neve teljesen: January
- d                 - egy/két számjegyes nap: 1
- dd                - két számkegyes nap: 01
- ddd               - nap neve rövidítve: Sun
- dddd              - nap neve teljesen: Sunday
- h                 - egy/két számjegyes óra: 0
- hh                - két számjegyes óra: 00
- n                 - egy/két számjegyes perc: 0
- nn                - két számjegyes perc: 00
- s                 - egy/két számjegyes másodperc: 0
- ss                - két számjegyes másodperc: 00
```

Alapértelmezett értéke: mm-dd-yyyy hh.nn

- maxthreads: a thearedek (szálak) előjelei. Ha majd a másolást külön szálak végzik, akkor itt maximalizálhatjuk a szálak számát. Jelenleg ez a változó határozza meg az egyszerre egymás mellett megnyitott nézőkék számát is. Alapértelmezett értéke: 5
- copybytes: Ha file-t másolunk (természetesen könyvtár másolás esetén a könyvtárban található file-ok is másolódnak), akkor annak mekkora részleteit olvassuk illetve írjuk egyszerre. Ennek a változónak az állítgatásával kioptimalizálhatjuk a sebességet. A mértékegység byte-ban értendő. Nincs megkötés az értékre, de a beállítóablakban csak a 10 és 100000 byte közötti értékek vannak elfogadva. Az alapértelmezett értéke: 32765
- fontheight: A betűk magassága a főablakban található összes elemnek (a többi ablakra

nem vonatkozik). Vigyázat nem méret, hanem magasság (a kettő nem ugyanaz). Alapértelmezett értéke: 12

- fontweight: A főablakban szereplő elemek betűinek a vastagsága. Ez sajnos nem lineáris, kb. 65 körül van a vékony és vastag között a határ (ez betűtípusonként különböző lehet). Alapértelmezett értéke: 70 (azaz vastag)
- viewerfontheight: A nézőkének a betű magassága. Alapértelmezett értéke: 12
- rowheight: a panelek sorainak a magassága. Szerintem ennek jelenleg nincs jelentősége, mert a fontheight paraméter alapján számítódik (+5). Alapértelmezett értéke: 18
- dist: ez az egyik legtöbbet használt elem. Szinte az összes formon (ablakon) szerepet játszik, és az elemek távolságát határozza meg. Alapértelmezett értéke: 2
- topshift: ez egy különleges változó. Egyes operációs rendszerek alatt a panelek egyes sorában a betűk el lehetnek csúszva. Ezekben az esetekben ezzel a változóval korrigálhatunk. Alapértelmezett értéke: 2 (nálam így van pozícióban)
- ignorecase: keresés esetben (file keresés illetve tartalomban keresés (nézőke is)) az összehasonlítás érzékeny legyen-e a kis-, nagybetű különbségre. Az „ALT”-os keresés az minden esetben ignorecase. Alapértelmezett értéke: true (azaz nem érzékeny)
- directory_mode: a filelistában a könyvtárak jobb elkülönítésére határothatunk meg a könyvtárak nevére egy külön formátumot. A formátumstringben a %DIR% fogja jelenteni az eredeti filenevet. Alapértelmezett értéke: %DIR%. Konfigurációs file-beli alapértelmezett értéke: [%DIR%]
- humanread: a filelistában a file-ok méretének megjelenési módja. humanread=true esetén a méret rövid formában jelenik meg az adott mértékegységgel (ha nincs, akkor byte). False esetben a méret byte-ban jelenik meg mértékegység nélkül. Alapértelmezett értéke: false, de a konfigurációs file-beli alapértelemezett értéke: true
- autocomplete: az inputsorok automatikus kiegészítése (parancssor és az ActionForm inputsoraira vonatkozik). Néha zavaró, de néha nem. Alapértelmezett értéke: false
- dirstr: ez az a string, ami a könyvtárak mellett a méret oszlopban jelenik meg, akkor amikor még nem néztük meg a méretét. Alapértelmezett értéke: Dir, de a konfigurációs file-ban <DIR>
- extls: régi funkcióját elvesztve, az új funkciója az, hogy a tömörített állományokban való navigálás során könyvtárba/ból lépés esetén történjen-e újra olvasása a tömörített állománynak. Ha beállítjuk, akkor csak akkor történik olvasatás, amikor a tömörített állományba belépünk, ez nyilván sokkal gyorsabb a navigálás közben. Alapértelmezett

értéke: true

A [color] nagyrész változói (alapértelmezett értékkel):

- col_panel_bg=clBase: a panelek hátterének színe (dark esetén fekete)
- col_panel_font=clBlack: a panelek betűinek a színe
- col_panel_exec=clBlue: a futtatható file-okra vonatkozó sorban levő betűk színe
- col_panel_hl_bg=clHighlight: a kurzor háttér színe (ez lesz a könyvtár pozíciót jelző rész hátterének is a színe)
- col_panel_hl_font=clHighlightText: a kurzor alatti sor betűinek a színe
- col_panel_hl_exec=clAqua: a kurzor alatti futtatható file-hoz tartozó sor betűinek a színe
- col_panel_sel=clRed: kijelölt sorok betűinek a színe
- col_cl_bg=clBase: a parancssor hátterének a színe
- col_cl_font=clBlack: parancssor betűinek a színe
- col_button_bg=clBackground: a főablakon levő gombok színe
- col_button_font=clBlack: a főablakon levő gombok betűinek a színe
- col_bg=clBackground: a főablak hátterének a színe
- col_font=clBlack: a főablak betűinek a színe

A [package] nagyrész (a beállítóablaknál már olvashattuk ugyanezt)

Az itt szereplő hozzárendelések rendelkeznek névvel, ami a tulajdonságok előtt találhatóak meg, és az első megjelenéskor automatikusan létrejönnek. Ezek alapján egy sor felépítése az alábbi: <név>_<tulajdonság>=<érték>. A nevet mi választhatjuk az tulajdonságok és értékeik az alábbiak lehetnek:

- ext: a hozzárendelés alapja, egy kiterjesztés (string)
- handle: a kezelő program/script neve. Ez található meg a pack/ (packageprefix) könyvtárban (string)
- copyin: a használt tömörítő program lehetőséget ad-e a tömörített állományba való másolásra (azaz tömöríteni) (bool)
- copyout: a használt tömörítő program lehetőséget ad-e a tömörített állományból való kimásolásra (azaz kicsomagolni) (bool)
- rm: törlési lehetőség (bool)
- dir: az adott program milyen file listát szolgáltat. Ha a filelista magában foglalja a könyvtárakat is és azok összes elérési útját, akkor lehet true, különben false. Ha nem

vagyunk biztosak a dolgunkban, akkor legyen false, mert ez általánosabban lehet jó. Ace, bz2, gz, rar, tgz lehet true tesztelés alapján, de zip, deb, rpm szigorúan false lehet. (bool)

- run: van e lehetőség file-t futattni az állományon belül. Ez csak deb, rpm és hasonló csomagoknál lehet hasznos, ahol így a csomagba lépve lehetősg nyílik telepítésre/frissítésre.

Nézzünk egy példát, ahol .ace-hez akarunk egy kezelőt csinaálni. A kezelő neve legyen ace, a kiterjesztés .ace, kezelőprogram uace (a pack/-ban, ez egy stript), a script csak listazni tud (alapkövetelmény), kimásolni és a filelista tartalmazza a fileok elérési útját:

```
ace_ext=.ace
ace_handle=uace
ace_copyin=false
ace_copyout=true
ace_rm=false
ace_dir=true
ace_run=false
```

Fontos megjegyezni, ha valamelyik tulajdonság hiányzik a konfigurációs file-ból akkor az vagy false lesz bool esetén, vagy üres string esetén.
Hívás:
%HANDLER% copyout %ARCHIVEFILENAME% % A_FILE_IN_THE_ARCHIVEFILE%
%TO_COPYFILE%

Az [extensions] nagyrész (a beállítóablaknál már olvashattuk ugyanezt)
Ha a panelen egy adott kiterjesztésű filera duplaklikkelünk vagy entert ütünk, és az nem tömörített file, akkor ezt a programot indítja el paraméterül átadva azt a file-t, amin entert ütöttünk.
Ennek a résznek a felépítése a következő: két részből áll egy hozzárendelés, egy kiterjesztésből és egy kezelőből. Legyen a hozzárendelés neve avi, ekkor a kiterjesztés rész az alábbi: avi_ext=.avi, a kezelő rész: avi_handle=xterm -e mplayer. Ezzel megadtam, hogy az .avi kiterjesztésű file-okhoz az mplayert indítsa.

Ezzel ki is tárgyaltuk a fő konfigurációs file-t és annak felépítését. A változók rossz alrészbe sorolása katasztrofális lehet. A nagyrészeket a filevége illetve egy újabb nagyrész definiciója fejezi be.

3. A fejlesztői dokumentáció

Sokan azt hiszik, hogy egy ilyen file-kezelő programot írni nagyon egyszerű: feldob két file listát, azután megy minden magától. Nos aki már nekivágott egy ilyen vagy hasonló programnak, vagy csak közelebbről kíséri végig az egyes „commanderek” fejlődését (például egy másik commander írója, hogy tudja, hol tart a konkurencia), akkor az nagyon jól tudja, hogy ez egyáltalán nem így van. Kezdve azzal, hogy valóban lehet találni file lista komponenst, de ennek használata a file böngészésben ki is merül, látható, hogy ha valaki komolyabb programot akar írni, akkor alapjaitól kell azt megírni, vagy a legegyszerűbb komponenseket választani, melyeket szinte tetszőlegesen lehet alakítani, formálni.
Egy file-kezelő a „két panel leple alatt” igen bonyolult struktúrákat vonultathat fel. Mint minden általános program esetén is rengetek kivételt kell lekezelni, amik esetlegesen csak hosszas tesztelés után derülhetnek ki. (De ez már a szoftver életútjának a problémája.)

3.1. Szerkezeti áttekintés

Talál már sejthető a felhasználói dokumentációból, hogy a sok lehetőség sok és kusza struktúrákkal van megvalósítva. Ezeket több módszer szerint is lehetne osztályozni, de az eredmény mindenféleképpen valami áttekinthetetlen rendszer lenne. Nos ennek ellenére megpróbálom összeszedni a megfelelő struktúrákat, úgy hogy azok összefüggéseire is fény derüljön. Nézzük a konkrét tényeket.
A legnagyobb osztályozási szempontnak azt a tényt venném, hogy a rendszer mely részei láthatóak, és melyek nem. Azok a részek melyek láthatóak, egymástól jól elkülöníthetőek, és a rendszer belsejét eléggé jól elrejtik a felhasználó elöl. Ezek a részek a fejlesztő környezet beépített komponensei. Ez jelen esetben a Borland Qt-je, mely nem túl designos, de a funkcióinak kifejezetten megfelel. (A designt illetően megoszlanak a vélemények, valaki a csicsás felületért van oda, de én fontosabbnak tartom a belsőt, sok-sok commander van már piacon melyek ugyan szép kezelőfelülettel rendelkeznek, de gyakorlatilag semmire sem használhatók.) Ezek kezelőelemek a gombok, listák, comboboxok, formok, scrollbarok, headercontrolok, menük stb. Ezekkel a programozónak nem sok dolga van, hiszen saját beépített függvényeik illetve tagváltozóik vannak, melyeken keresztül kontrolálhatjuk ezeket.
A másik rész az, ami a felhasználók elöl rejtve marad, amit a programozó leprogramoz. A

háttér, ami a munkát elvégzi, amikor a felhasználó valamit változtat a kezelőfelületen keresztül. Ez a rész teremt kapcsolatot a tényleges fizikai szervezéssel, azaz a file-okkal és a látható felület között. Amennyire tőlem tellett én megpróbáltam ezt az elkülönítést több kevesebb sikerrel.

3.2. A belső

A belső felépítéssel kezdeném, mert a külső rész magyarázatához, annak funkcióihoz a belső rész ismerete szükséges. Próbálom az alapoktól felépíteni az egymásra épülő strukturákat.

3.2.1. A CMyList osztály

Ez egy dinamikus template alapú lista, mely jól kezelhető. (Itt jegyeznem meg az egyik alapvető problémát, hogy bár van ilyen beépített C++ struktúra, de mikor ezeket inculdoltam a program nem volt hajlandó lefordulni. Ezért vált szükségessé egy saját dinamikus lista létrehozása.)
Alapvetően ez egy felspécizett kétirányú láncolt lista, mely alapvetően az osztályt felhasználó programozó rejtve marad, és csak a tagfüggvényeken keresztül érheti el, vagy pedig tömbként is kezelheti, azaz az egész struktúra indexelhető. A lista elemei, a template-ből fakadóan, tetszőleges osztály lehet.
A felhasználás során a MyList.h header file include-olása után, az alábbi módon deklarálhatunk változókat ezzel a típussal: „CMyList<AnsiString> *dirs;” Ekkor a lista elemei AnsiString típusúak lesznek (ez egy jól kezelhető string megvalósító osztály, sok helyen ezt használom, ha string típus kell. Számtalan beépített függvénnyel rendelkezik.).
A CMyList osztály alap típusa az element, mely a láncolt lista egy eleme. Ez tartalmazza az adatot, plusz a két pointert, az előző és a következő elemre.
A CMyList osztály beépített tagváltozói:
Az összes változója private. Van két element típusú pointer, melyek a lista elejére (first) illetve végére (last) mutatnak, valamint még egy int típusú paraméter, mely a lista hossza (len).
A CMyList osztály beépített tagfüggvényei:
Ezek public tulajdonságúak.

- A konstruktor(): Ez felel azért, hogy létrejöjjön egy üres sor. Az üres sornak a hossza 0, az eleje és a vége pedig NULL-pointer. (Általában igaz az, hogy ha az eleje NULL, akkor már a vége is).
- A DelList(): Ez egy létező listát töröl ki. Ha valamelyik az eleje és a vége közül NULL, akkor az szükségszerűen hozza a magával a másik NULL-ságát is, persze a biztonság kedvéért beállítom ezeket az értékeket NULL-ra. Ha létezett listaelem, azaz egyik sem volt NULL, akkor egyessével kitöröljük az elemeket, majd a listát is, a len-t meg 0-ra állítom.
- AddFirst(T): Ez paraméterben megkapott adatot a lista elejére szúrja be. A pointerek beállítása külön macerás, mert még esetlegesen a last-tal is foglalkozni kell az első elem esetén, de inkább nekem így jobb, mintha fejelemes listát használnék (a felhasználónak nem úgyis mindegy).
- AddLast(T): Hasonló, mint az AddFirst, csak ebben az esetben a lista végére szúrunk, hasonlóan szigorú megkötések mellett.
- GetFirst() és GetLast(): ezek az első illetve utolsó elemmel térnek vissza. Leellenőrzi, hogy van-e ilyen ha nincs, akkor a default konstruktor által meghatározottal tér vissza.
- DelFirst() és DelLast(): az első illetve utolsó elemeket törli ki, megfeleő pointer ellenőzésekkel. (Az előbbi három függvényosztály lehetőséget ad, a FIFO illetve stack típusok kezelésére.)
- Get(int=0): Ez visszatér a paraméterben megkapott elemmel (tömb tulajdonság). Ha nincs paraméter megadva a függvény hívásakor, akkor az első elemmel tér vissza (tömb analógiában a 0-dikkal). Fontos, ha túlindexelünk, akkor az utolsóval tér vissza.
- Del(int=0): Ezzel tetszőleges elemet lehet törölni. Ha nem adunk paramétert, akkor az első elem kerül törlésre, márha létezik egyáltalán. Túlindexelés esetén, az utolsó eleme törlődik.
- IsNull(): bool visszatérési értékében adja meg, hogy üres-e a lista, ehhez a first és last pointereket használja fel.
- Lenght(): a lista hosszát adja meg. (A default lista hossza 0.)
- [int] operator: Ez az operátor biztosítja számunkra, hogy a listát tömbként indexelhessük. Ha nem referenciával kell visszatérni (jobbérték), akkor kapásból csak a Get(int) függvényt hívja meg. Balérték használata esetén, hasonló eljárás fut le, de üres lista esetén, exception dobódik, nyilván ekkor nem lehet referenciát létrehozni.

3.2.2. A CRPMLikePackage osztály

Ez az osztály teszi majd lehetővé, hogy azokat a csomagokat kezeljük, melyek nem relatív hivatkozással adják meg a fileok helyét, hanem abszolúttal, esetlegesen azoknál is jó, ahol a könyvtárak nem kapnak külön entry-t. Talán nem látszik elsőre, de ez azoknál is működik, akiknél alapból nem alkalmazzuk. Tehát ez egy bővebb halmazt fed le, így bizonytalanság esetén érdemes ezt használni. Mellesleg a felhasználó ezt a tulajdonságát a csomagnak a vc.cfg konfigurációs file [package] részében a dir tulajdonság határázza meg egy-egy csomagkezelőnél. Ha ez false, akkor az bizony RPMLikePackage lesz.
Az egész egy speciális, butított lista lesz, ami arra jó, hogy egy elemet csak akkor vesz hozzá a listához, ha az még nincs benne.
A tagváltozók (private):

- dirs: Ez egy AnsiString-gel példányosított CMyList lesz.
- sp: a stack pointere. (Ez majdnem felesleges, de régen kellett mikor még nem volt dinamikus listám.)

A tagfüggvények (public):

- A konstruktor(): sp-t 0-ra állítja és létrehozza a default listát (üres lista).
- A destruktor(): ez a listát kitörli.
- Add(AnsiString): Ez először leellenőrzi, hogy a beszúrandó string benne van-e már a listában, ha igen false-szal tér vissza, jelezve, hogy már volt egy ilyen elem. Ha nem volt, akkor az első helyre beszúrjuk azt, és megnöveljük a sp értékét. Mindegy lenne, hogy a list elejére vagy végére szúrunk be, de általában az azonos könyvtárral rendelkező állományok egy helyen vannak, így az elejére beszúrva, sok ellenőrzéstől kíméljük meg magunkat, hisz a lista elejéről keresve kapásból rá lelhetünk az előzőleg beszúrt azonos elemre.
- GetSp(): az sp értékével tér vissza.
- static GetNextDir(AnsiString ad, AnsiString pn, bool &id): Ezzel a pn-ben található elérési útvonalban az ad-vel kezdődő, majd az utáni elemet adjuk, meg. Ekkor vagy ez az utolsó elem amivel visszatérünk (isdir ekkor false), ha ez egy közbülsö elem lesz, akkor isdir értéke true lesz, azaz bisztosan egy könyvtár, mert utána még jönnek elemek. Ha az ad nem prefixe pn-nek vagy egyéb kivételes eset van, akkor üres sztringgel térünk vissza.

3.2.3. A CMyFtp osztály

Nos ez az osztály felel az FTP kapcsolatért. Jelenleg igen gyenge állapotban van, és a gyengeségeit magasabb szinten egyenlítem ki. Számtalan függvényt tartalmaz, melyekbe itt nem megyünk bele, mert szorosabban nem képzi részét egy ilyen filekezelőnek. A függnányek listája magyarázatokkal:

- Connect: FTP kapcsolatot hoz létre.
- Login: bejelentkezés a szerverre (felhasználónév és jelszó alapján).
- Dir: könyvtárlistát ad egy megadott file-ba
- Get: file-t tölt le a szerverről
- MkDir: könyvtárat hoz létre a szerveren (megfelelő jogok esetén).
- Put: file-t tölt föl a szerverre (megfelelő jogok esetén).
- Delete: file-t töröl a szerverről (megfelelő jogok esetén).
- RmDir: üres könyvtárat töröl az FTP szerverről (megfelelő jogok esetén).
- Noop: Noop-ot küld (ezzel ellenőrzöm egy kapcsolat meglétét).

Ez egy eléggé buta osztály, mert találtam olyan FTP-servert, mellyel nem volt képes kommunikálni. Így az osztály felújításra szorul.

3.2.4. A CMyBookmark osztály

Ez az osztály személyesít meg egy bookmark elemet. Érdekes lehet, hogy míg ez látható a menüben is, miért kell neki külön struktúra. Ez azért van, mert mikor létrejön egy menübejegyzés, akkor mindegy egyes menüsornak külön kezelő függvénye van, ha ráklikkelnek. Ennek az osztálynak az elsődleges szempontja, hogy a menüsorokhoz biztosítsa a kezelő függvényt.

A tagváltozók:

- mi: ez egy TMenuItem típusú változóra mutató elem. Ez azért csak pointer, mert a menü struktúrában már egyszer létezni fog, így érthetően mi azzal akarunk majd dolgozni.

A tagfüggvények:

- A default konstruktor: ez csak úgy van, de igazából sehol sincs használva, mert az mi változó NULL lesz.
- A konstruktor(TMenuItem): ez a bejegyzést úgy hozza létre, hogy már előre tudjuk a hozzá tartozó menüelemet. Ekkor az mi ezt az értéket fogja felvenni, és az mi OnClick

függvényét az osztály BookmarkMenuClick függvényére állítom, hogy a menüponton való klikkelést én kezelhessem le.
- BookmarkMenuClick: Erre hárul az a feladat, hogy a menüponton való kattintásra a panel oda ugorjon, ahova a menüpont mutat.

3.2.5. Olyan struktúrák, melyeknek nincsenek tagfüggvényei

A CPac osztály

Ez az osztály szükséges ahhoz, hogy a tömörített osztályokat kezelni tudjuk. Ez fogja hordozni, azokat az információkat, melyek az egyes kezelőkhöz tartoznak.
A tagváltozók:
- Name: a kezelő neve.
- Ext: melyik kiterjesztésre legyen ez az aktív kezelő.
- Handle: a kezelő scriptjének a neve, mely ténylegesen meghívásra kerül.
- copyin: lehetséges-e a tömörített fileba belemásolni egy állományt (jelenleg a könyvtárakra is ez vonatkozik, köszönhetően a legtöbb tömörítő script nem rendelkezik mkdir opcióval).
- copyout: lehetséges-e az adott file-ból kicsomagolni egy másik file-t, a kimásoló rutin el fogja végezni, az alkönyvtárak és azok tartalmának másolását is (ellentétben a betömörítéssel).
- rm: törlési lehetőség a tömörített állományban.
- dir: a script milyen filelistát állít elő. Ha vannak külön könyvtárak kezelve, és az útvonalak relatívak, akkor lehet true. Ellenkező vagy ismeretlen esetben false.
- run: A tömöritett állományban futtatási joggal rendelkező file-ok indíthatósága a scripten keresztül. (Ez kifejezetten fontos az RPM-nél és a deb-nél csomagoknál fontos, ahol lehetőség nyílik azok telepíthetőségére ezen a változón keresztül.)
- listed: Ha az állította be a felhasználó a programban, hogy ő csak a tömörített állományba való belépéskor szeretné kilistáztatni a tartalmát, akkor ebben a változóban tartom nyílván, hogy ez megtörtént-e már.
- num: a száma a kezelőnek.

Az osztály a Main.h-ban kerül deklarálásra, és a Main.cpp-ben tartozik hozzá néhány kezelő függvény. Ezek az alábbiak:

- GetPacInd(AnsiString): ha kezelő név alapján szeretnénk megállapítani a hozzá tartozó tulajdonságokat. Azaz egy csak egy indexszel tér vissza.
- GetPacIndByExt(AnsiString): ugyanaz, mint az előző, csak most kiterjesztés alapján határozzuk meg az indexet.

Ezek a függvények fixen a CMyList<CPac> pacs; változóban keresnek, ez a Main.h-ban található TForm1 osztály deklarációjának része.

A CExt osztály

Ez rendkívűl hasonló a CPac osztályhoz, de annál egyszerűbb. A tagváltozói:

- Name: a kezelő neve.
- Ext: a kezelő melyik kiterjesztéshez tartozik.
- Handle: a kezelő program neve (szükség esetén útvonallal).
- num: a kezelő száma.

Ez is a Main.h-ban van deklarálva, és a Main.cpp-ben tartozik hozzá néhány függvény is:

- GetExtInd(AnsiString): ha kezelő név alapján szeretnénk megállapítani a hozzá tartozó tulajdonságokat. Azaz egy csak egy indexszel tér vissza.
- GetExtIndByExt(AnsiString): ugyanaz, mint az előző, csak most kiterjesztés alapján határozzuk meg az indexet.

Ezek a függvények fixen a CMyList<CExt> exts; változóban keresnek, ez a Main.h-ban található TForm1 osztály deklarációjának része.

A CSearch osztály

Ez egy igen elavult speciális lista osztály, ami még használatban van, és az előtt került használatba, hogy a CMyList-et létrehoztam volna. Ez egy láncolt lista, melybe a keresés során kerülnek a file-ok. Eltárolásra kerülnek az alábbi értékek:

- dir: a file-hoz tartózó elérési út.
- num: az elérési út könyvtárában az adott file hányadik bejegyzés. (Furcsa lehet, hogy miért nem a nevét tároljuk, nem tudom, így jött ki a lépés. Természetesen ha időközben megváltozik a struktúra, akkor más file kerülhet az adott bejegyzésszámhoz. Ez legtöbbször nem okoz problémát.)
- next és prev: a lista következő és előző elemei.

Az osztály használata csak a keresésnél fordul elő. Így ott még találkozunk vele.

A CDirHist osztály

A könyvtárváltáskori történet követést tesz lehetővé, ha ezt az osztályt listába szervezzük. Váltáskor a lényegi álapotokról felvételt készíthet, melyeket később vissza lehet állítani. Jelenleg, csak akkor használatos, amikor tömöritett állománybeli másik tömöritett állományba lépünk. Ezt csak úgy tudjuk megtenni, ha egy temporális területre kitömörítjük azt.

A tagváltozók:

- dir: a tömörített állomány neve elérési útvonallal.
- archdir: hol voltunk ebben a tömörített állományban.
- mode: milyen módban voltunk a váltás pillanatában (jelenleg csak ARCH lehet).
- def_pac: mi volt az előző állapothoz tartozó csomagkezelő, azaz milyen típusú tömörített állományban voltunk.

A my_file osztály

A név konvenció hiányából is látszik, hogy ezt az osztályt már a kezdetek óta használom. Az legalapabb háttérben tartózkodó file-lista egy eleme ilyen típusú lesz. Azaz ez teénylegesen egy fizikai file-t jelent. Ez a rész rejtve marad a felhasználó elöl, de a látható panel az adatokat innen veszi, illetve jeleníti meg. Rengeteg tagváltozóval rendelkezik, nézzük ezeket sorra:

- Name: a file neve, ez jelenik meg a kijelzőn névként. A SEARCH mód esetén ez tartalmazza az útvonalat is.
- Mask: ez az elnevezés még nagyon régről ered, és már nem is azt jelenti, amit régen, de én már hozzászoktam. Ez tárolja a filehoz tartozó jogot egy stringként. Nem igényelnek a jogok különösebb struktúrát, mert így is jól kezelhetőek. a szokásos dugsrwxrwxrwx.
    1. d – ha könyvtár, l- ha link (könyvtárra mutató link esetén is)
    2. uid bit (u)
    3. gid bit (g)
    4. sticky bit (s)
    5-7. a tulajdonos bitjei (olvasás, írás, futtatás) (rwx)

8-10.a csoportok bitjei (rwx)

11-13. az egyebek bitjei (rwx)

- Size: a file mérete (__int64, mert a manapság használatos nagy file-ok (2G-nél nagyobbak) sok gondot jelentettek a sima int-nek)
- Date: a már formázott file-hoz tartozó módosítási dátum.
- Type: a file típusa. Könyvtár esetén „Directory”, egyéb esetben a file attributuma (ami egy szam, de itt most stringgé konvertálódik).
- Link: ha a file szimbolikus link, akkor ez modja meg hova mutat, amúgy üres.
- ext: a file kiterjesztése '.'-tal együtt.
- pref: a file neve útvonal nélkül (NORMAL módban ugyanaz, mint a Name).
- ii: beszédes név arra utal, hogy kiíratásnál milyen ikonja legyen.
- Focused: az adott file aktív-e (szerintem sehol nincs használva).
- Selected: Ez már annál inkább fontosabb, mert ez mondja meg, egy file ki van-e jelölve.
- Owner és Group: a file tulajdonosának neve és csoportja stringben (nem használt)
- uid és gid: a file tulajdonosának neve és csoportja számmal (igen fontos)
- exec: megmondja, hogy a file futtatható-e (a jogokból van számolva, de elég csak egyszer meghatározni, ne kelljek a kiíratónak is tudni ezt a számolási módszert).
- stat: a file attributuma, de itt most számmal (nem használt).

A file-lista feltöltése majd a CAct osztályban lesz részletesen tárgyalva.

3.2.6. A CMyDirent osztály

Miből lesz a file-lista. Ez az osztály felel a file-ok tulajdonságainak meghatározásáért. Ez az osztály teremt kapcsolatot a tényleges fizikai fileszervezés és a file-lista között. egy ilyen típusu elem mindig egy adott könyvtár tartalát és annak tulajdonságait hordozza.

A private tagváltozói:

- location: az a könyvtár elérési úttal, melyről az információkat hordozza. (Nem létező esetén marad a régi, ha ilyen nem volt, akkor NULL.)
- prevlocation: Az előző elérés, azért fontos, hogy rossz file-lista olvasás esetén legyen hova visszatérni. A lista frissítése után, meg fog egyezni a locationnal, mert tovább nem lesz szükség az előző könyvtárra, csak a következő listafrissítésnél.
- count: a file-ok száma a könyvtárban.

- dir: ez a DIR* típusú változó mely lehetővé fogja tenni a könyvtár bejegyzéseinek végig olvasását.
- ent: ez egy struct dirent* típus. Ebbe kerül bele egy bejegyzés, amit azután feldolgozunk részletesebben (lényegi információja csak a d_name mezője, ami a bejegyzés, azaz file illetve alkönyvtár neve).

A public tagváltozója:

- filelist: ez a my_file-ra emlékeztető struktúra lista, csak annál egy kicsit butább, és jobban ki van hegyezve a valós file tulajdonságaira. A mask mezője hasonló mint a my_file esetében, de itt a könyvtárra mutató link is d-vel kezdődik ellentétben a my_file Mask mezőjével, ahol az l-lel kezdődött ilyen esetekben. A date mező itt int típusú, később ez kerül formázásra a dateformat alapján (lásd konfigurációs file).

A tagfüggvényei:

- A konstruktor: count 0, filelist NULL, prevlocation, location dummy értékekkel rendelkeznek.
- Location(): visszatér az aktuális location értékkel.
- Count(): az aktuális bejegyzés szám értékkel tér vissza.
- SetLocation(AnsiString): beállít egy új elérési útvonalat, majd ahhoz frissíti is a file-listát. Sikeres frissítés esetén true-val tér vissza, sikertelen esetben false-szal. Lessük meg ezt egy kicsit alaposabban. Ha üres új útvonallal hívjuk meg, akkor „/” lesz az új elérési útvonal. Egyéb esetekben is kiegészíti az új útvonalat '/'-rel, ha az nem erre végződne. A bejegyzések elolvasásához a dir-be nyitja meg az elérési útvonalat, ha ez nem sikerült, az egész false-szal tér vissza. Siker esetén először megszámolja a bejegyzéseket azért, hogy a filelist-nek lefoglalhassa a megfelelő mennyiségű helyet a memóriában. Ezt meg is teszi (természetesen az előző listát törli). Ez követően újra veszi a bejegyzéseket, melyek alapján a struct stat64 típusú tf változóba bele is rakja az egyes bejegyzések tulajdonságait. Ez fölhasználva kitölti a filelist tömb aktuális mezőjét. Ehhez alapvetően az lstat64 függvényt használja, mely a linkek esetében magáról a linkről ad infót nem a referált file-ról. Ezután ráeresztjuk a readlink függvényt mely megmondja, hogy szimbolikus link e az adott file, ha igen, akkor a stat64 függvénnyel megállapítjuk a referált file tulajdonságát is, és a linktargetet is. A mask meghatározása külön trükk, mert egyesével meg kell nézni az egyes tulajdonságokat (bővebben ezekről a tulajdonságokról „man 2 stat” parancsal érhetünk el). A bejegyzések végigolvasásával lezárjuk az adott könyvtárat.

3.2.7. A CMyThread osztály

Ez az osztály egy kezdeményezés, de ennek ellenére már meglehetősen nagyra duzadt. Alapvető célja ennek az osztálynak, hogy azokat a műveleteket elkülönítse a többiektől, melyeket párhuzamosítani lehet. Jelenleg most a három alapműveletnek nevezhető műveletet tartalmazza: másolás, mozgatás és törlés. Még ezek nem is futnak igazából párhuzamosan, de minimális átírással megtehető lenne. A legnagyobb probléma ez ellen, hogy egyes esetekben hajlamos lefagyni a párhuzamos műveletvégzés közben. Ezért most még ez az osztály csak egy hely,ahol néhány művelet össze van szedve.

A párhuzamosítás egyik lényeges része, hogy a globális változók kezelését kerülni kell. Ezért ez az osztály sok olyan tagváltozót alkalmaz, melyek globálisan is megtalálhatóak lennének, de azok módosítása beláthatatlan következményekkel járna. A számtalan változó mellett sok tagfüggvény is van, melyekre a műveletek jól szét vannak bontva. Nézzük a tagváltozókat és azok funkcióit:

- panel: ez tartalmazza a file-listát melynek az elemeinek egy részén kell majd a műveleteket elvégezni (itt lesz szükség a my_file::Selected-re).
- count: a tömb mérete.
- srcdir és dstdir: mivel a művelet elején egy „pillanatképet” készítünk a jelenlegi állapotról itt határozzuk meg a forrást és célt (legtöbbször a panelek könyvtára ez, de változtatni is lehet).
- whattodo: a művelet neve.
- mask: azt a maszkot határozza meg, hogy a file-ok melyeken a műveletet végezzük milyen céleredményt produkáljanak (már ha van cél).
- mode: a forrás panel milyen módban van. (a párhuzamos műveletek, leginkább csak NORMAL és SEARCH módban működnek).
- symlinks és attr: az attribútumok megőrzéséhez illetve a szimbolikus linkek követéséhez tartozó kapcsolók (a felhasználó állíthatja ezeket az ActionForm-nál).
- copybytes: az egyszerre másolandó byte-ok száma (konfigurációs file-oknál bővebben).
- allproceed, allproceed_err, allproceed_app: esetleges hiba, vagy cél létezése esetén a felhasználó választhatja, hogy nem kíván foglalkozni a hasonló témájú kérdésekkel, ekkor ez az értéket billentheti be.
- progform: a művelethez tartozó ProgressForm, azaz az a form, ami a műveletet állapotát

jelzi, hogy az éppen hol tart.

- created: létre lett-e hozzva a thread.

A tagfüggvények:

- Create: Ez a függvény legyen a legelső, amit létrehozunk még azelőtt mielőtt elindítanánk Execute-tal a műveletet. Itt készíthetünk pillanatfelvételt a rendszerről. Sok paraméterrel rendelkezik, de csak így lehet jól szeparáltan futtathatni. Természetesen a hívása előtt még sok-sok ellenőrzésre van szükség, de ezt majd a megfelelő helyen (CAct osztálynál).
- Destroy: megsemmisíti a threadet. Jelenleg semmi haszna sincsen.
- Execute: csak létező threadre hívható meg, különben kilép. Ez csak a todo() függvényt hívja meg, majd annak lefutása után a threadet megsemmisíti, hogy ugyanide új thread jöhessen létre.
- todo(): a Create-nél meghatározott műveletet hajtja végre.

Most jöjjenek a művelethez szorosan kapcsolódó fűggvények:

- ChmodFile_my: egy adott file-ra beállítja a beépített chmod függvény segítségével a jogokat, és annak hibaüzeneteit is lekezeli.
- IsDir: jelenleg nem használt függvény, előzetesen megállapította valamiről, hogy az könyvtár-e. TÖRLENDŐ!
- GetStringByMask(AnsiString s,AnsiString m): az s stringből az m maszk alapján egy újabb stringet hoz létre. (Lásd a felhasználói dokumentáció egyszerűsített maszk részét.) Ekkor a maszk lesz visszatérő string, de ha abban '*' szerepel, akkor oda behelyettesíti az s stringet.
- CopyFileTo_Base(AnsiString Src,AnsiString Dst,__int64 size): Ez az én általam megírt másolás függvény, mely az Src-ben meghatározott egyetlen file-t átmásolja a Dst-ben meghatározottra. ha a Dst létezik, akkor ahhoz, hozzá lesz fűzve az új file. Miért? Azért mert a cél létezésének a vizsgálatát, már korábban el kell végezni, és ott eldönteni, hogy mi legyen az eredmény: felülírás (ekkor törölni kell), hozzáfűzés (nem kell csinálni semmit), ne történjen másolás (ekkor el sem jutunk eddig a függvényig). A CopyFileTo_Base függvény úgy lett kitalálva, hogy kezelni tudja a PrograssFormot is, úgy hogy a hátralevő időt becsülni is tudja. A file-ok megnyitását a fopen64-gyel végzem, mert sok a nagy (2G<) file, amit a fopen nem képes lekezelni. Az Src file-ból fread-del olvasunk copybytes-nyit, majd amennyit sikerült olvasni, annyit fwrite-tal ki is írunk a Dst-be. Hiba esetén azt is lekezeljük, és false-szal tér vissza a függvény. Minden egyes olvasás esetén ha eltelt egy másodperc, akkor sebességet és hátralevő időt

számolunk. Végül lezárjuk a file-okat és true-val tér vissza.

- A front-end függvények (Copy_my(), Move_my(), Delete_my()): Ezeket a függvényeket hívja mef a todo(). Ezekben a függvényekben hajtjuk végre a műveleteket, azokon a file-okon melyek a panel file-listában ki lettek jelölve. Elvégzi a ProgressForm beállításait, majd végig mely a listán, és ha file-t talál, akkor CopyFileTo_my/ MoveFileTo_my/ DeleteFile_my-t hívja meg, ha szimbolikus linket, akkor azt külön lekezeli, ha könyvtárat, akkor a CopyDir/ MoveDir/ DeleteDir függvényeket hívja meg. Külön figyel arra, hogy a másolás az NORMAL vagy SEARCH módú panelról történik-e. Másolás illetve mozgatás esetén prefix vizsgálat is történik, egy egyszerű AnsiString::Pos függvény hívással. Hiba esetén a szokásos formmal találkozhatunk. A művelet végén bezárja a ProgressFormot.
- CopyDir, CopyFileTo_my: a CopyDir akkor hívódik meg, amikor könyvtárat kell másolni a tartalmával együtt, így gyakra rekurzívan hívódik meg. Első lépésnek megézi, hogy be tudott-e lépni a forrás könyvtárba, ha nem akkor 1-gyel tér vissza a függvény (hibátlan befejezés esetén 0-val kellene). Ha sikerült, akkor továbblépvén a célt hozzuk létre. Majd a könyvtár összes bejegyzésén végigmegyünk és átmásoljuk. Itt megint csak lehetnek szimbolikus linkek, könyvtárak és sima file-ok. Ezek kezelése hasonló, mint a Copy_my-nál tárgyaltaknál. Szimbolikus linkek kezelése a symlinks bittől is függ, amit még afelhasználó állíthatott be a másolás esetén. Előfordulhat az is, hogy a felhasználó a ProgressFormon Cancelt nyomott, ekkor ezt is lekezeli a másolás (és a többi művelet is). A CopyFileTo_my már csak egy file másolását végzi el, de úgy, hogy előtte a célt is leellenőrzi, hogy az létezi-e már, és ha igen, akkor megteszi a megfelelő intézkedéseket, a felhasználó akaratának megfelelően. Végül meghívja a CopyFileTo_Base-t, ami a tényleges másolást végzi. Ennek sikere esetén beállítodnak a célra megfelelő jogok, ha a felhasználó ezt beállította. Hiba esetén itt jelenik meg az az ablak, melynél a felhasználó reagálhat a hibára.
- MoveDir, DeleteDir, MoveFileTo_my, DeleteFile_my: a mozgatás leginkább a törlésre hasonlít, de sok szempontból a másolásra is. Először is a *Dir esetekben nem elegendő végigmenni a könyvtár bejegyzésein, mert azok esetlegesen már megsemmisültek. Így itt más módszert kell alkalmazni mint a másolás esetén. Azt teszem, hogy mindig az első file/könyvtárat törlöm, ha az nem sikerült, akkor növelek egy számlálót és továbbá a szerint fogok törölni. Minden sikertelen törlés után növelni kell a számlálót, hogy az egyszer nem tudott törölni file-okat többször ne próbáljam meg letörölni. A

végelszámolásban a növelt érték mennyisége határozza meg, hogy az egész hibás lett-e. Fontos megjegyezni, hogy a mozgatásnál, ha file-okat nézzük és nem sikerült azt mozgatni (pl.: jogi probléma, vagy különbözőek a partíciók), akkor először másolja, majd törölni próbálja a file-t (a törlés szintén nem feltétlenül sikerül jogi problémák miatt).

Ezzel nagyjából le is zárhatjuk ezt az osztályt, de fontos felhívni a figyelmet néhány buktatóra. Egyik a NORMAL és SEARCH módú panelek kezelése közötti különbség. Másik a szimbolikus linkek kezelésének mássága a többi file-tól. A hibák kezelésére is külön figyelni kell.

3.2.8. A CAct osztály

Ennek az osztálynak CMyPanel is lehetne a neve. Ez az utolsó bástya a háttér- és a látható rész között. Minden mást ez csinál a háttérben, amit az eddigi osztályok nem (például: FTP-s másolás, vagy tömörített állományok kezelése). Alapvetően két változó fog létezni ebből a típusból, mindkét látható panelhez egy-egy. A közös funkciókat, melyek mindkét panelra hatnak, a főablak osztálya fogja elvégezni. Ez a legnagyobb osztály (bár csak kicsivel, de megelőzi a Main.cpp-t is, ami a főablakhoz tartozik), így számtalan funkciót lát el. Sok kapcsolat található itt a vizuális elemekkel is, mert így a tényleges látható elemek amúgy kusza állapotát így lehet kompenzálni, mert jól el lesz különítve az egyes panelokhoz tartozó elemek. A panelek közül mindig az egyik aktív, a másik (a többi, jelenleg egy) a passzív (nem aktív), a főablaknál ezt az act illetve nact változókkal jelölöm (ezek típusa a CAct). Így bárhonnan hivatkozva el tudom érni bármelyik panel bármelyik elemét. Nézzük ennek az osztálynak a tagváltozóit.

A látható részekhez tartozó változók:

- active: ez egy pointer a ténylegesen látható file-listára. Ennek típusa TDrawGrid, mely megengedi a grafikák megjelenítését a szövegek mellett. Ez az ikonok miatt rendkívül fontos. Szinte ez az egyetlen típus ahol a sorok alakíthatóságára teljes a szabadság. (A headercontrol nem része a listának, persze a kettő szorosan összefügg.)
- dirloc: ez a panel felett található rész, mely a könyvtár pozíciót mutatja. Én is ennek a paraméternek a Text mezője alapján döntöm el hol vagyok. Ez máshol tárolásra nem kerül. Változtatva a Text mezőt, maga a megjelenített is meg fog változni.
- sb: a panel melletti scrollbar-ra mutat. A TDrawGrig beépített scrollja meglehetősen gyenge, ezért az ki van kapcsolva és ez van helyette.

- selbar: a file-lista elemeinek kijelölés összegző státusz jelzője ez. Különösebb szerepe itt nincs. (Szerintem a pointer felesleges rá!)
- free: annak a partticiónak az állapotárol ad információt, ahol az adott könyvtár található.

Most következzen a nem látható részekhez kapcsolódó változók:

- num: a panel innen tudja magárol, hogy ő most melyik is a sok közül.
- prevdir: ez mindig az előző könyvtár ahonnan jöttünk. Abból a célból kell, hogy a kurzort jó helyre pozicionáljam frissítéskor. Ebből következik, hogy egy könyvtárba való belépéskor ez az érték „..”.
- selected: a kijelölt file-ok száma. Fontos a selbar szempontjából is (statisztika), de kiemelt szerepet játszik, a CheckSelected() függvény esetében is, mivel mindig csak kijelöl file-okon tudunk műveletet végezni.
- selected_bytes: (__int64) a kijelölt byte-ok száma, pusztán statisztikai célokból.
- sum és sum_bytes: ugyanaz, mint az előző esetben, de itt kizárólag statisztikai célokkal a panelben található összes file-ra kijelöléstől függetlenül.
- mode: a file-lista módja, azaz hogy éppen hol is vagyunk. NORMAL (0, szokványos lista), ARCH (1, tömörített állományban vagyunk), FTP (2, FTP-lista), SEARCH (3, keresés eredménye jelenik meg) módjai lehetnek. A műveletvégzés szempontjából rendkívüli fontossággal bír, hiszem más-más alrutint kell meghívni különböző esetekben (már ha az adott módban ez lehetséges).
- sortmode: a panelben a file-ok milyen szempont szerint legyenek rendezve: 0, ha név szerint (alap), 1, ha méret szerint, 2, ha dátum szerint.
- sortdir: a rendezés iránya: 0, ha növekvő (alap), 1, ha csökkenő.
- archdir: talán amikor azt írtam, hogy a dirloc->Text-ben van az ahol vagyunk, akkor nem mondtam teljesen igazat, mert belépve egy tömörített állományba, már nem azt jelöli, ez csak a file nevéig jelez, de hogy azon belül hol vagyunk azt ez a változó határozza meg. FTP módban is ez módja meg hol vagyunk, de ekkor ezt hozzáadom a dirloc->Text-hez is. (ARCH módban technikai okok miatt nem, megvalósítható lenne, de így most jó!)
- search, newsearch: a keresésnél játszanak szerepet. Ez lesz a CSearch típusú láncolt lista (kezdemény) változói. search-ben lesznek eltárolva a file-ok pozíciói. A newsearch pedig egy techinai változó, hogy a listán végig tudjunk menni.
- search_num: az adott listában elhelyezett elemek száma. (A panel tömb méretének a meghatározásához kell.)
- def_pac: ha ARCH módban vagyunk, akkor ez a változó mondja meg, hogy éppen milyen

típusú a tömörítés (rar, gz, zip stb.).

- ftp, ftpbuf: az FTP-zéshez szükséges technikai változók.
- allproceed, allproceed_err: a belső műveletek végzése során a felhasználó döntéseit tároló változók (esetleges hiba, vagy váratlan esemény eldöntésére illetve annak automatizálására).
- servername, prevuserpass: az előzőleg használt FTP szervernek a neve, illetve az ahhoz használt felhasználónév és jelszó (egyben '/'-rel elválasztva). A szerver neve tartalmazhatja a port számot is (ftp.bmw.hu:21).
- act: ő az erre az objektumra mutató pointer (azaz a this). Technikai okok miatt kell, egyszerűsíti az átírást, és jobban elkülöníti a tagváltozókat is.
- mrt: a multirename tool ablaka lesz, amikor szükség van rá.
- actionform: az actionform ablaka lesz amikor szükség van rá. (Ez a két ablak modális mindig, mert egy-egy felhasználói döntést reprezentálnak, így belőlük egy is elég.)
- dirhist: jelenleg csak a tömörített állományban levő tömörített állományba lépés esetén követi az utat, de tetszőleges is lehetne. Tehát könyvtár történetet tart nyílván.

Ezzel nagyjából fel is soroltuk az összes változót. Következzenek a tagfüggvények:

- a konstruktor: a változók értelemszerű alapértékeinek beállítása, különös tekintettel az egyes pointerek értékeire. (Külön itt nem magyarázom.)
- RefreshPanel(): nos ezzel a függvénnyel lehet újraolvastatni a panel tartalmát, ez persze a SEARCH módban nem érvényes, és a ARCH mód extls bit igaz értéke mellett sem, de a többi esetben igen. Az egyes módok részeinek a kezelése természetesen eltérő lehet, de az alapvető séma ugyanaz, az alaplépések megegyeznek:
    - A dirloc->Text és a archdir változók alapján meghatározott hely bejegyzéseinek újraolvasása:
        - NORMAL: dirent->SetLocation függvényével, hiba esetén egy szinttel feljebb lép (ha a hiba a '/'-ben történt, akkor szerintem elszáll, de ezt elvileg mindig tudjuk olvasni, ha nem, akkor úgyis mindegy).
        - ARCH: az extls és az aktuális tömörítési eljárás listed változója alapján dönti azt el, hogy kell-e újraolvasni. Ha igen, akkor meghívja a tömörítési eljáráshoz tartozó kezelő scriptet, majd a listed értéket true-ra állítja ezzel jelezve, hogy a listázás megtörtént. A stript kimenete egyenlőre a temporális könyvtár pack file-jába lesz kiirányítva, azaz itt kell megjelennie a formázott file-listának.

    - FTP: ez is csinál egy listát az ftp->Dir-rel, ami még másképpen lesz formázva mint, ahogyan az később szükséges lesz. Ha itt hiba történ, mert már esetleg nincs meg az ftp kapcsolat, akkor megkérdezi, a felhasználót, hogy akar-e újracsatlakozni. Ezután lefut a formázó script az előzetesen létrehozott ({tmpdir}/ftp) file-ra, ami ugyanoda, azaz a temporális tárterületre létrehozza a ftp2 file-t. A temporális terület alatt itt most azt a könyvtárt kell érteni, amit az alábbi módon határozunk meg: tmppath+"fctmp"+uid
    - SEARCH: itt mint korábban említettem nincsen új lista kérése.
- Ezeket után jön a panel előzetes elemeinek kitörlése, és az RPM típusú csomagok kezeléséhez szükséges könyvtár lista (dirlist) létrehozása.
- Meg kell határozni az új panelben hány darab file lesz:
    - NORMAL: a dirent->Count() fuggvénye pont ezt határozza meg.
    - ARCH: végig kell olvasni, a pack file-t, majd annak a megfelelően formázott soraira számolni, abban az esetben, amikor jó könyvtárszinten vagyunk. RPM típusú csomagoknál még azt is meg kell nézni, hogy az adott könyvtár szerepelt-e már, ha nem, akkor szintén számolni kell, különben nem. A számolás az arch_count változóban kerül eltárolásra.
    - SEARCH: itt könnyebb a dolgunk, hiszen a search_num-ban benne van a találatok száma, már csak ezt kell megnövelni egyel (a „..” miatt).
    - FTP: az ftp2 file-on kell végig menni, ez is egyszerűbb, mint az ARCH eset, mert itt a lista csak egy könyvtárra vonatkozik, nem az egész ftp site-ra. Hiba esetén visszalépünk NORMAL módba. A számolás eredménye itt is az arch_count változóban tárolódik el.
- Az eddig az RPM típusú csomagoknál használt dirlist-et kinullázzuk, hogy reprodukálni tudjuk, a számlálásnáli környezetet. Majd a kiválasztásokat is megszüntetjük, a számlálókat pedig kinullázzuk.
- Ez után jön a tényleges panelbeli lista előállítása (ez még mindig nem látható):
    - NORMAL: dirent->Count()-nyi helyen a dirent->filelist elemeit felelteti meg a panel megfelelő értékeinek. Dátum meghatározódik a formátum alapján. Az ikon száma hozzárendelődik a kiterjesztés alapján, a sum és sum_bytes változók megnövekednek a megfelelő értékkel. Ha a könyvtár egyben link is, akkor a mask első mezője 'd'-ről 'l'-re változik. Ha az adott

név megegyezik a prevdir-rel akkor az adott sorszám eltárolásra kerül (! rendezésnél fontos szerepet játszik majd).

- ARCH: természetesen a pack file végigolvasása alapján határozza meg a panel tömb elemeinek megfelelő értékét. Csak néhány érték lesz figyelembe véve a tömörített állományok szűkített használata miatt. Az első bejegyzés speciális, mert ez a „..”-ot fogja létrehozni, azaz, hogy mindig képesek legyünk egy szinttel feljebb lépni illetve a tömörített állományból kilépni, akár olyan esetekben is amikor az valamilyen oknál fogva nem tartalmaz bejegyzéseket, vagy csak hibásakat tartalmaz. Az egyes sorokat ':'-ok alapján bontja szét három oszlopra: jogok, méret és név. A névvel sokat kell trükközni az útvonal tartalma miatt! Ez természetesen típusonként eltérő lehet. Hasonlóan a NORMAL módhoz itt is van prevdir ellenőrzés, azaz könyvtár váltás esetén jó pozíciót kapjunk.
- SEARCH: Itt érdekes módon határozzuk meg a panel egyes elemeit. A keresés során a file-okhoz csak azok könyvtárát illetve abban elfoglalt bejegyzési helyét tároljuk el. Ezen információk alapján kell megállapítani a tényleges eredményt. Ha belegondolunk ez nem is nehéz. Végigolvassuk a search-listát (search/newsearch), ha itt új könyvtárral találkozunk, akkor beolvassuk, különben hogy gyorsabb legyen, nem olvassuk be mégegyszer ha az előzővel megegyezett. Az első elem itt is kiemelt szerepet játszik, ez a „..”.
- FTP: teljesen hasonló az ARCH módhoz, mert itt is file-ból (ftp2) olvassuk be a bejegyzéseket, de itt mindig csak egy könyvtárra vonatkoztatva, így a feldolgozás nem annyira bonyolult. Az első elem itt is a „..”. Hiba esetén visszaugrunk NORMAL módba.

- Ezután jön a rendezés, de ennek csak NORMAL, FTP, ARCH módban van értelme, de SEARCH módban nagyon nincs ezért ebben a módban nem is lesz ez érvényben. A rendezést a panelsort függvény végzi, a panel tömbön, aminek meg kell adni a hosszát, és az aktív sorát, hogy az a rendezés során ne keveredjen el.
- a rendezést követi, hogy a látható panelek sorainak a számát meghatározzuk. A scrollbar hozzáigazítjuk ehhez a mennyiséghez.
- Ha „..” nem legfelül lenne, mert a rendezés során elkeveredett, akkor ezután ezt a lista legtetejére pakoljuk. Végül az aktív sort kiválasztjuk. És a parancssor melletti

útvonalat az aktuális panel könyvtárpozíciójához igazítjuk. A panelek alatti információs részeket is frissítjük, majd az RPM típusú csomagokhoz használt dirlist-et teljesen kitöröljük.

- Copy_my: Ez a függvény felel a másolás előtti döntések lebonyolításáért: shift lenyomása esetén helyben másolás, tömörített illetve ftp esetben a megfelelő függvény meghívása. NORMAL mód esetén ez a függvény kezeli le azt az ablakot, mely megkérdezi a felhasználótól a célt, a maszkot illetve a többi beállításokat. Ezután leellenőrzi, hogy van e kiválasztás, amire a műveletet érvényesíteni kell, ha nincs akkor az aktuális file-t választja ki. Ezután a létrehozza a megfelelő threadet végül elindítja azt. A művelet után frissíti a panelokat.
- Move_my: ez mar csak NORMAL-NORMAL panelok esetén működik. Szokásos kérdés, hogy hova és milyen néven, majd kijelölés ellenőrzés, végül végrehajtás (CMyThread).
- MkDir_my: FTP mód esetén meghívja a megfelelő függvényt, egyéb esetben csak NORMAL módban megy. Megkérdezi a könyvtár nevét, majd létrehozza azt.
- Delete_my: ARCH és FTP módokban meghívja a megfelelő függvényt, NORMAL és SEARCH módban megkérdezi, hogy tényleg le akarjuk-e törölni a kijelölteket (vagy az aktívat). Ezután le is törli a kívánt file-okat. Ha a panel SEARCH módban volt törlés előtt és nem ignoráltuk a kérést, akkor letörli ugyan a file-okat, de törlés után visszavált NORMAL módba (ez ugye a már korában is tárgyalt SEARCH nem-frissítése miatt van).
- Enter_my(): Ez a függvény hívódik meg minden esetben, amikor a panelen entert ütünk. Ha az aktuális elem a „..”, akkor a cdpp() függvény hívódik meg. Különben leellenőrzi, hogy amin állunk az „Directory” típusú-e, ha igen, akkor abba belelép. ARCH és FTP módokban az archdir módosul, míg NORMAL módban a dirloc->Text az ellenőrzés után, hogy sikerül-e az új könyvtárba belépni. Ezt a futtathatóság tulajdonságával (exec) nézik meg. Ha valamilyen oknál fogva nem sikerült belépni, vagy a tartalmat olvasni, akkor visszatérünk a régi könyvtárhoz. Ha az aktív elem nem könyvtár, akkor a következő menet alapján dől el, hogy mi legyen. 1. kiterjesztés alapján megnézi, hogy ehhez a kiterjesztéshez van-e kezelő, ha van, akkor FTP mód esetén kilép, a többi mód esetén ARCH módra vált, és beállítja a def_pac-ot a kiterjesztéshez tartozó indexre. (ARCH mód esetén előtte még kitömöríti a file-t és elraktározza a dirhist-be, hogy honnan jött.) Ha nincs a kiterjesztéshez csomagkezelő, akkor megnézzük, hogy van-e egyéb program hozzárendelve ehhez a kiterjesztéshez. Ha igen, akkor azt végrehajtjuk. Ha nem, akkor

végső fázisként megnézzük, hogy futtatható-e az állomány, ha igen, akkor elindítjuk (ARCH esetén megnézzük a speciális run bitet is).

- Rename_my: Csak NORMAL módban működik. A szokásos ablak az új névhez. Ha van kijelölve file, akkor '*' jelenik meg célnévnek, ha nincs kijelölés, akkor az aktuális file neve.
- CheckDP(int): a paraméterben megadott sorról megnézi, hogy az „..” e vagy sem.
- RefreshSelBar(): ebben a függvényben frissítem a panel alatti információs részt. Kiírja az összes file számát és méretét, és a kijelöltekét is. A méretet ha kell, átalakítja mértékegységessé a jobb olvashatóság kedvéért. Ezenkívül, az adott partícióról is kapunk információkat: méret, szabad hely, ezek százalékos aránya, a partíció mount helye. Majd frissíti a hozzátartozó kijelzőket is (selbar és free).
- CheckSelected(): Ez fogja leellenőrizni, hogy van-e kijelölés. Ha nincs, akkor az aktuális file-t kijelöli.
- SelectAll(): Az elnevezés megtévesztő, mert ez az invert selection. Azaz végigmegy a listán és ha ki volt jelölve, akkor leveszi a jelölést, egyébként rárakja.
- GetDirSize(AnsiString): a paraméterben meghatározott könyvtár méretével tér vissza. Természetesen az alkönyvtárakat is beleszámolva (rekurzívan), akkor ha az nem szimbolikus link.
- Create_my: egy file létrehozásának a környezete ez. A FileCreate függvénnyel hozza létre a file-t. Sikertelenség esetén tájékoztatja a felhasználót. Az inputban megadott file-név mindig az aktuális panel könyvtárpozíciójához viszonyítva relatívan jelenik meg.
- CreateSymlink_my: egy szimbolikus link létrehozásának a környezetét biztosítja, azaz leellenőrizi a módot, megkérdezi, hogy a link hova mutasson és hol legyen. Csak NORMAL módban lehet linket létrehozni, és egyenlőre csak egyet (technikai okok miatt nem lehet többet).
- EditSymlink_my: A társa az előzőnek, azaz ha rossz linket csináltunk ezzel tudjuk szerkeszteni. NORMAL és SEARCH módban is.
- Copy_Arch: ezzel a függvénnyel el is érkeztünk a „feketeleveshez”. Ez egy átfogó függvény, mely arra szolgál, hogy eldöntse, hogy a panelek közül melyik van ARCH módban és melyik NORMAL-ban, mert a program jelenleg, csak vagy NORMAL -> ARCH vagy ARCH -> NORMAL másolást tud csinálni. Megnézi, hogy a megfelelő eset fennállása esetén van-e a tömörítés kezelő scriptnek copyin illetve copyout paramétere.
- Copy_FromArch: ez a tömörített állományból való másolást oldja meg. Sajnos itt egyes

file-okra vonatkoztatva nem tudjuk megmondani, hogy hol tartunk, csak azt látjuk, hogy melyik file van éppen tömörítés alatt. A tömörített állományból való kimásolás menete a következő:

1. Végigmegyünk a kiválasztási listán.
2. Ha könyvtár tulajdonságúval találkozunk, akkor újból el kezdjuk olvasni a már RefreshPanel-nál is tárgyalt „pack” file-t, amiben még mindig az adott tömörített állományra vonatkozó információk vannak eltárolva. Ebből kiszedjük azokat a bejegyzéseket, mely az archdir+”a mostani könyvtár ahol tarunk”-kal kezdődnek, azaz azokat a bejegyzéseket, melyek az adott könyvtár alatt helyezkednek el tetszőleges mélységgel.
3. Ha nem könyvtár, hanem sima file, akkor az standard (a script copyout paraméterével, természetesen) másolásra kerül.

- Copy_ToArch: Ez már kevésbé megerőltető, mert itt nem teszünk különbséget könyvtár és file között, hanem ezek kezelését 100%-ig a scriptre bízzuk. Erre azért volt „szükség”, mert egyes tömörítések ezt kezeli, egyesek pedig nem. A biztos persze az lenne, ha mi csinálnánk meg, de ekkor még az is előfordulhat, hogy a tömörítés nem kezeli a könyvtárlétrehozást a tömörített állományon belül. A másolás menete csak annyi, hogy végigmegyünk a kijelölt file-okon, és meghívjuk erre a kezelő scriptet (copyin).
- MkDirLong_my(AnsiString, AnsiString): hosszú könyvtárat hoz létre (2. paraméter) az 1. paraméterben megadott helyre, tehát annak léteznie kell. Sajnos a MkDir beépített függvény csak egy szint mélységig tudja kezelni. Így a '/'-ek mentén kell vagdosni a stringet.
- View_my(): a file-tartalom nézegetéséért felel. Leellenőrizi, hogy van-e szabad viewform a maxthreads-nyi közül. Ha nincs, kilép, ha van, akkor azzal fog dolgozni tovább. FTP módban az 1 MB-nál kisebb file-ok letöltésre kerülnek, a nagyobbakat nem lesz lehetőség megnézni. Könyvtár esetében a már tárgyalt GetDirSize függvényt hívja meg. Az ARCH módban a megnézendő file-t kicsomagolja a script copyout paraméterével. Így az összes módban használható a nézőke. Ha extviewer változónak van értéke, akkor a külső nézőkét hívja meg, ha nincs, akkor a belsőt (csak belső nézés esetén (SHIFT+F3) más függvény kerül meghívásra).
- ChmodDir_my, ChmodFile_my, Chmod_my: Ez a függvényosztály a jogok beállítását teszi lehetővé külön file-ra és külön könyvtárra. A Chmod_my a kiinduló függvény, ez kezeli azt az ablakot, melyben a felhasználó meghatározza az adni kívánt jogokat. Ezt

abszolút módon teheti meg. Ezután megtörténik a szokásos kijelölésvizsgálat, majd a kijelölt file-okon végigmegy. Csak NORMAL és SEARCH módban működik. Ha könyvtárt talál és be volt kapcsolva a rekurzivitás, akkor jut szerephez a ChmodDir_my, meg előbb magára érvényesíti az új jogokat, majd a tartalmára is (rekurzívan). A tényleges jogbeállítást file-okra és könyvtárakra egyaránt a ChmodFile_my végzi, mely sikertelen jogbeállítás esetén egy hibaüzenettel ajándékozza meg a felhasználót.

- ChownDir_my, ChownFile_my, Chown_my: A file-ok tulajdonosát és csoportját változtathatjuk meg. Ez teljesen olyan, mint a Chmod, de itt csak egy sorban kell megadni a felhasználói id-t és a csoport id-t is ':'-tal elválasztva. Ugyanaz a beállítás lesz érvényben könyvtárakra és file-okra is.
- SelectMask(): a maszk általi kiválasztást hajtja végre. A feldobott ablakban lehet beállítani a kívánt maszkot, majd végigmegy a file-okon és kijelöli azokat melyekre a FitOnMask igazat ad vissza. Ha az ki volt már előtte is jelölve, akkor annak a file-nak a kijelölése meg fog szűnni. Legvégül a RefreshSelBar-ral frissítjük a selbart.
- SelectNone(): ez csupán annyit tesz, hogy végig megy a panel listán és kijelölt file-okról leveszi a kijelölést. Ellenben a többi commanderrel, itt a könyvtárakra is ugyanúgy érvényesek a kijelölési szabályok, mint a file-okra. A kijelölések után nem kell frissíteni a panel tartalmát, mert az ilyenkor nem változik, elegendő csak a megjelenítőt frissíteni (act->active->Invalidate()).
- SelectNNone(): Ez egy iszonyat szerencsétlen elnevezés, mert ez lenne a SelectAll, de azt már korábban lefoglaltam, így maradt a SelectNone ellentéte. Ez kijelöl minden file-t előzetes kijelöléstől függetlenül.
- FindDir, FindFile_my, FindInFile: Ez a függvényosztály file-ok keresésére szolgál, a file nevére illeszkedő maszk és a file tartalmára illeszkedő maszk szerint. Ennek az osztálynak a kiinduló függvénye a FindFile_my. Itt tudja majd a felhasználó meghatározni a név- és a tartalom maszkot. Beállítja a módot SEARCH-re, majd meghívja a rekurzív FindDir függvényt az éppen aktuális könyvtárra. Ez a függvény végigmegy a paraméterében megkapott könyvtár bejegyzésein. Ha könyvtárat talál és be van kapcsolva a rekurzív mód, akkor erre a könyvtárra hívja meg saját magát (!vigyázat szimbolikus linkek miatt végtelen ciklusba kerülhet). File esetén megnézi, hogy a neve illeszkedik-e a névmaszkra, majd a tartalommaszkra, ha igen, akkor felveszi a search láncolt listába és a search_num értékét is megnöveli. Ejtsünk néhány szót a FinfInFile-ról is:

  Ez a függvény fogja eldönteni, hogy a tartalom illeszkedik-e majd a maszkra.

Itt nem csak egy szimpla mintaillesztési feladat van, hanem maszk alapjáni. A file-t karakterről karakterre olvassuk, majd illesztjük a maszkra, csillag esetén könnyű a dolgunk, mert csak el kell tárolni ennek a pozicíoját mind a maszkban, mind a tartalomban és továbblépni a maszkban. Ha nincs tovább maszk, akkor igazzal kell visszatérni. Az új-sor karaktert átalakítjuk egy egyszerű üreskarakterré, majd megnézzük, hogy figyelmen kívül kell-e hagyni a kis- és nagybetűk közötti különbséget. Ezután nézhetjük a tényleges egyezőséget. Igaz esetben továbblépünk a maszkban, ha annak vége és a filenak is, akkor jó, ha csak a maszknak van vége és nem volt csillag sehol, akkor nincs illeszkedés, ha volt csillag, akkor ahoz az állapothoz térünk vissza, ahol azt találtuk. Ha a maszk és a tartalomban olvasott karakter nem egyezett meg, akkor megint csak az utolsó csillag dönt, azaz ha volt akkor oda térünk vissza, ha nem volt, akkor az illesztés eredménytelen lett. Ezután mindenféleképpen a tartalomban előre kell haladni. Elképzelhető, hogy az algoritmus nem tökéletes, de az esetek nagy részében jó. Ez a rész esetlegesen még tesztelésre szorul.

- CheckUpDp(int): leellenőrizi, hogy a listában legfölül a pont van-e. Ha nincs akkor azt odarakja, úgy hogy a paraméterben megadott elem pozícióját is nyomon tudjuk követni.
- ShowAllDirectorySize(): az adott könyvtárban az össze alkönyvtár méretét megmutatja a GetDirSize függvénnyel.
- Delete_FromArch(): tömöritett állományból töröl, ha ARCH módban vagyunk, és a tömörítés támogatja a törlést, azaz az „rm” opció igaz. Csak egy megerősítést kér, majd a kijelölt fileokra meghívja a kezelő scriptet az „rm” paraméterrel. Nincs megkülönböztetve a könyvtár és a file, ezt a tömörítési eljárásnak támogatnia kell.
- NewFtpConnection(): ezzel elérkeztünk az FTP rész tárgyalásához, amit szívem szerint kihagynék (mert eléggé hiányos szegénykém), de most inkább legyen. Ablakban megkérdezi, a szerver nevét, a kívánt csatlakozási portot, a felhasználónevet és a hozzátartozó jelszót. Ezek megadása után ezeket ellenőrzi, majd csatlakozik és belogol (ftp->Connect illetve ftp->Login). Tetszőleges hiba esetén tájékoztatja a felhasználót, majd kilép. Siker esetén beállítja a módot FTP-re, végül frissíti a panelt.
- ftp_reread(AnsiString): a paraméternem megadott FTP-könyvtárat frissíti az ftp2 file-ba, hogy azzal majd dolgozni lehessen, majd visszatér a bejegyzések számával.
- ftp_getname: régebben csak a file nevével tért vissza, most már az összes fontosabb

tulajdonságával is, az ftp2 file alapján. Szükség esetén újraolvassa az FTP könyvtárat. Névvel, attribútummal, mérettel tér vissza paraméterekben, amúgy igazzal ha sikerült meghatározni, false-szal ha nem. A file meghatározása úgy történik, mint keresés esetben, azaz csak azt mondjuk meg, hogy melyik könyvtárban van és ott hányadik bejegyzés (ez egyértelműen azonosítja a file-t).

- ftp_isdir: egy FTP file-ról eldönti, hogy könyvtár-e (szerintem felesleges, mert az ftp_getname attr paraméteréből ez meghatározható).
- MkDir_myFtp(): az FTP-szerveren való könyvtárlétrehozás környezetét teremti meg. Ablakban megkérdezi a felhasználót, hogy mi legyen a név, majd azt az ftp->MkDir függvénnyel megpróbálja létrehozni. Hiba esetén a felhasználó tájékoztatást kap.
- Copy_Ftp: ez is hasonlóan egy összefoglaló függvény, mint ARCH módban a Copy_Arch. Itt dől el, hogy melyik panel az FTP-s panel, azaz, hogy feltöltéshez vagy letöltéshez való függvényt kell-e meghívni.
- CopyDir_FromFtp, CopyFile_FromFtp, Copy_FromFtp: ez a letöltéshez szükséges függvényosztály. A Copy_FromFtp a kiindulási pont (a Copy_Ftp által is meghívott), itt kerül elő az elfogadáshoz és a cél megállapításához szükséges ablak (ActionForm). Majd a kijelölt elemeket is ez kezeli. Könyvtár esetén a CopyDir_FromFtp-t hívja meg (rekurzív), file esetén a CopyFile_FromFtp-t. A CopyDir_FromFtp felhasználja az ftp_reread függvényt, hogy meghatározza a másolandó file-okat. Ezeken végig megy könyvtár esetén újra hívja magát. A CopyFile_FromFtp egy file-t tölt le az ftp->Get függvénnyel. Felülírásra és hibára is figyelmezteti a felhasználót.
- CopyDir_ToFtp, CopyFile_ToFtp, Copy_ToFtp: ezek a függvények kezelik a filefeltöltést. Legelőször ellenőrzi, hogy az FTP-kapcsolat létezik-e még, ha nem kilép. Itt is megkérdezi a célt, ellenőrzi a kijelöléseket, majd végig megy azokon. Könyvtár esetén a CopyDir_ToFtp hívódik meg mely előbb megprobálja a könyvtárat létrehozni a szerveren, majd a tartalamkat is feltölti. File esetén a CopyFile_ToFtp hívódik meg. Ez az ftp->Put-ot fogja használni a feltöltéshez. Hiba esetén figyelmezteti a felhasználót.
- DeleteDir_my_Ftp, Delete_my_Ftp: FTP-szerverről törli a kijelölt file-okat és könyvtárakat. A könyvárak törlése az ftp->Rmdir-rel történik, de csak üres könyvtárra, ezért szükséges előtte annak tartalmát is törölni, könyvtár esetén rekurzívan. File törlése az ftp->Delete-vel történik. Itt is szükséges nyilvántartani, hogy hány file-t nem sikerült törölni, mert azokkal többet már nem kell próbálkozni.
- CheckFtpConnection(int): a ellenőrzi, hogy létezik-e a kapcsolat vagy sem, ha nem akkor

meghívja a NewFtpConnection függvényt. A paraméter az határozza meg, hogy tudjuk-e már, hogy nincs kapcsolat (0, ha tudjuk). A kapcsolat meglétének az ellenőrzését noop küldéssel tudjuk megnézni.

- GetFreeSpace: a dirloc->Text-hez tartozó partícióról szerez információkat egy speciális scripttel, ami a „packageprefix” változóban meghatározott könyvtárban található és dinfo névre hallgat. És a temporális könyvtár dinfo file-jába menti el az információt, amit később feldolgoz ez a függvény. Szükséges információkat paraméterben adja vissza: free (szabadhely), total (partíció mérete), per (foglaltság %-ban), phere (a partíció mounthelye).
- cdpp(): ez a függvény az Enter_my()-nál is meghívódik, ha a „..”-on ütöttünk entert (meg még ki tudja mikor). Nézzük eges módokban mi is történik ekkor:
    - ARCH: ebben a módban az archdir dönti el, hogy mi fog történni (ez mondja meg, hogy hol vagyunk éppen az tömörített file-ban). Ha ez nem üres, akkor ebben az archdir-ben egy '/'-rel visszább menyünk, majd újraolvasatunk. Ha ez üres, akkor kilépünk a tömörített állományból, először is megnézzük, hogy a dirhist (itt követjük nyomon, hogy honnan jöttünk) üres-e, ha igen akkor visszalépünk NORMAL módba majd tovább lépünk (azaz a többi lépést már a NORMAL résznél tárgyaljuk). Ha a dirhist nem üres, akkor már egy másik tömörített állományból jöttünk, tehát oda kell visszalépni, majd a dirhistből ki kell szedni ezt az elemet.
    - FTP: ennél a módnál is az archdir alapján döntünk, itt viszont a végét már a '/' jelzi, azaz ekkor leszünk az FTP gyökerében. Ebben az esetben fogunk elszakadni az FTP-szervertől, visszaváltunk NORMAL módba, a többi lépés már ott fog történni. Ha a szerveren mélyebben vagyunk, akkor egy '/'-rel vissza kell lépni és újraolvasatni.
    - SEARCH: ebben az esetben, ha „..”-tal találkozunk, akkor az mindenféleképpen a SEARCH mód elhagyását jelenti. Váltás NORMAL-ba, majd a többit ő intézi.
    - NORMAL: itt már a dirloc->Text lesz a főszereplő, mert ez határozza meg, hogy hol vagyunk. '/'-nél kijjebb nem mehetünk, mert az a gyökér. Amúgy egy '/'-rel visszalépünk. Fontos, hogy ekkor a visszatérési érték 0. Azokban az esetekben, amikor maga a cdpp csinálta a refrest, akkor a visszatérési érték 1.
- panelsort: ez a panel tömböt rendezi le, a rendezéshez beszúró rendezést használ, emellett megtartja az egyetlen kiválasztott elem pozícióját. A rendezési feltételt a sortmode

változó határozza meg, az irányt pedig a sortdir. (Más rendezési algoritmust is próbáltam, de nem hoztak javulást (qsort).)

- CloseFTPConnection(): lezár egy FTP-kapcsolatot, ha FTP módban voltunk. Az új mód NORMAL lesz. Az új könyvtár pedig a gyökér ('/').
- InternalView(): ugyanaz, mint a View_my(), de itt nincs külső program hívás.
- MultiRename: ez egy csak NORMAL módban működő több file különböző szabályok szerinti átnevezésére alkalmas függvény. Első lépésben létrehoz egy MultiRenameTool-t, majd ennek a listájába betölti a kiválasztott elemeket (file-okat). Ezek után a felhasználó kiválasztja a szabályokat (részletesebben később), majd „Ok" esetén végrehajtja az átnevezéseket.
- ClearSearch(): a search listát törli ki. Minden egyes új keresés esetén meghívódik.

Ezzel be is fejeztük a CAct osztály tárgyalását, ami azt jelenti, hogy lezárult a nem látható részek tárgyalása. Ezután jönnek azok az osztályok, melyek szorosan azokhoz a részekhez kapcsolódnak melyeket a felhasználó is lát. Ez nem jelenti azt, hogy ezek a részek nem tartalmaznak semmi hasznosat, csak a Kylix által legenerált kódot. Ezek a részek is bőven tartalmaznak algoritmusokat, melyek mind egy reakció a felhasználó által generált eseményekre.

3.3 A külső

A külsőt alapvető kinézetét a beépített Borland Qt komponensek határozzák meg. Ezeknek az elrendezése viszont teljesen a program általam leprogramozott része végzi, azaz nem hagyatkozom ilyen szempontokból az alapértelmezett elemekre. Ezeknek az elemeknek, javarészét a programból saját úton kezelem, igyekezvén a beépített funkciók számát a legkisebbre korlátozni (egyes funkciók hasznossága vitathatatlan és más úton megvalósíthatatlan lenne). A program elemeinek futásidejű elrendezése azért szükséges, mert egyes rendszerek meglepően sok különbséget képesek mutatni. Ilyen eset, amikor valamelyik betűtípus nincs az adott gépen, így azt valami mással helyettesíti, ami már nem ugyanakkora méretű. Ekkor az elemek elcsúsznának egymáshoz képest a statikus esetben. Másik eset amikor a felhasználó átméretezi az ablakot.
Nem szándékom leírni az egyes ablakok részletes tulajdonságait, hisz az szinte teljességgel lehetetlen. Megpróbálok azokra a dolgokra koncentrálni, melyek a felhasználói

dokumentációból nem derülnek ki, vagy melyek nem egyértelmű úton lettek megvalósítva.

3.3.1. A TForm1 osztály, avagy a főablak és kezelőelemei

A főablak egy olyan form, amely menüvel rendelkezik. Ezen kívül az alábbi tulajdonságokkal bír: átméretezhető keret (fbsSizeable), stílusa normális (fsNormal), alapértelmezett pozíció és méret (poDefault, ezzel lehetővé válik az induláskori futásidejű ablak pozíció és méret beállítása, amit az előző kilépéskor mentett el a program). A form méretére nincsnek megszorítások, szabadon átméretezhető. Részletesen az elemeket a dokumentáció elején található képen láthatjuk. Van néhány elem mely szorosan a láthatósághoz tartozik, de mégsem látszik. Ez a főablak felosztása, mely 4 fő részből áll: menü, panel1, panel2 és a parancssor része. A menü egyértelmű és csak a menüt tartalmazza, a panel1 a látható elemeket tartalmazza a lista felett levő gombokkal és információs ablakokkal, valamit egy másik panel elemmel, mely a headercontrol kezeléséért felel. A panel1 balra van rendezve, így lehetővé válik egy splitter beillesztése, mellyel a panelok arányát állíthatjuk be. A panel2 ugyanaz, mint a panel1, de ennek a rendezettsége már kliensfüggő, ez is szükséges a splitter helyes működéséhez. A panel3 a parancssort és az előtte levő útvonalat, valamint a mögötte levő 'X' gombot tartalmazza.
Ebben az osztályban találhatunk néhány fontos változót, melyeknek nem sok köze van a láthatósághoz, de igazából nem is panel specifikusak, ezért kerültek ide. Ezek a megvalósítás szempontjából jelentős szerepet játszanak. Nézzük ezeket:

- panels: ez egy két elemű tömb, mely a két panelt tartalmazza. Kevésszer lesz használva, mert nem tudjuk, hogy éppen melyik az aktív. Ehelyett leginkább ennek az elemeire mutató pointerek lesznek a fontosak (act illetve nact).
- readedconfig: a configfile beolvasottságára utalna, de csak az átméretezés során van használva.
- prate: a két panel közötti elválasztás helye. 0 és 1 közötti szám, alapból 0,5, azaz középen van. 0 felé az első panel lesz kisebb, 1 felé a második.
- pfind: az ALT+BETŰ hatására beírt szöveg itt tárolódik el.
- sprcss[], prcss_n: az elindított programokhoz tartozó processzek számai itt tárolódnak el, majd ez alapján lesznek megvizsgálva, hogy befejeződtek-e már, hogy ki lehessen őket lőni, mert ezek a processzek, igazából nem fejeződnek be a program befejeztével, hanem egy olyan állapotba kerülnek, amikor a szülő utasítására várnak. De az (a szülő, azaz a

VC) nem tudhatja, hogy az általa indított program mikor fejeződik be, így az a listából rendszeresen ellenőrizni kell (ez a rendszeresség jelenesetben a billentyűzet lenyomás).

- mybookmark: egy CMyBookmark-kal példányosított láncolt lista lesz. Tehát itt lesznek a függvények az egyes bookmark elemeken való kattintáshoz.

A változók után nézzük a fontosabb függvényeket:

- chk_pcss(): ez fogja kilőni a már befejeződött processzeket. végigmegy a sprcss[] listán, és a waitpid függvénnyel megnézi, hogy létezik-e egyáltalán a processz, ha nem, akkor kiveszi a listából. Ellenkező esetben megnézi annak státuszát, és ha már WIFEXITED, akkor szintén kiveszi a listából.
- CallKeyDown, CallKeyUp: ezek a függvények felelnek a billentyűzet kezeléséért. Minden látható elemnek saját gomblekezelő függvényei vannak, de hogy ezeket egy helyen kezeljem, így az összes ezt fogja meghívni. Nézzük mit is csinálnak ezek. A CallKeyUp nagyon egyszerű, mert ő csak azt vizsgálja, hogy a felengedett gomb ALT-e, ha igen, akkor a pfing stringet kinullázza. A CallKeyDown ennél lényegesen bonyolultabb:
    1. Lefuttatja a chk_pcss()-t, hogy a végzett processzektől megszabaduljunk.
    2. Leellenőrzi, hogy az ALT le van-e nyomva és emellett az új gomb is megfelel-e annak, hogy arra keresni lehessen. Ha igen, akkor azt hozzáadja a pfind stringhez, és megjeleníti a StatusBar1 kijelzőjén. Majd a listában rákeres, hogy van-e pfind prefixű sor. Ezután kilép.
    3. Most egy bonyolult ellenőrzés jön annak eldöntésére, hogy a parancssor kapja-e meg a vezérlést. Abban az esetben, ha olyan billentyűt nyomott le a felhasználó, hogy az egyértelműen a parancssorhoz tartozik (pl.: valamilyen betű), akkor egyből itt kezelődik le az esemény. Ha olyan billentyű került lenyomásra, mely a panelnál és a parancssornál is fontos, akkor a parancssor aktivitása dönti el, hogy ki kezelje le az eseményt (pl.: kurzor gomboknál). Azaz ilyen esetben ha a parancssor volt aktív, akkor az kezeli le, ha nem akkor a panelek. Ha a parancssor nem volt aktív, de a felhasználó leütött egy olyan karaktert, melyet hozzá kell adni a parancssorhoz, akkor annak inaktivitása miatt ezt nekünk kell megtenni, majd aktívvá tenni a parancssort. A backspace is hasonló eset. Ha nem a parancssor kezeli le az eseményt, akkor a következők történnek: Előfordul, hogy nem a panel az aktív, hanem éppen a parancssor, ekkor nekünk kell lekezelni az eseményt, ellenkezőleg a panel maga megcsinálja, majd aktívvá tesszük a panelt.

4. Ha még nem lépzünk ki, akkor meghívjuk a RefreshButtons függvényt, hogy kezelje le, azt amit eddig nem kezeltünk le.

- RefreshButtons: olyan billentyűlenyomásokat kezelünk le itt, amit máshol nem. Ezeket a felhasználó nem tudja megváltoztatni. Nézzük a feldolgozás menetét:
  - A CTRL és le/föl gombok leellenőrzése. Ha igen, akkor a parancssor története legördül. Ezután kilép.
  - TAB: ChPanel() függvény meghívása (lásd később). Ezután kilép.
  - ESC: törli a parancssor kiírását, majd kilép.
  - Enter: Ha meg volt nyomva a CTRL, akkor meghívja a Ctrl_Enter_my-t, majd kilép. Ha a SHIFT és az ALT is meg volt emellett (enter) nyomva, akkor meghívja a CAct::ShowAllDirectorySize() függvényt, az összes könyvtár méretének a megállapításához, majd kilép. Ezután ha még idáig is eljutottunk, akkor megnézi a parancssort. Ha az üres, akkor az Enter_my()-t hívja meg, ellenkező esetben a parancssor tartalmaz valamit, azaz azt végre kell hajtani. Ha ez cd-vel kezdődik, akkor saját kezelésben történik a parancssor elemzése, tekintettel, hogy NORMAL vagy FTP módban vagyunk e. Ha nem „cd”-vel kezdődött, akkor a parancs végrehajtásra kerül. Xshellben, ha az 'X' gomb be volt nyomva, vagy nélküle, ha nem. Ekkor az ExecCmd kerül meghívásra. Ha a parancssor tartalmaz '<'-t vagy '>'-t, akkor 'X' beállításától függetlenül nem fog xshellt indítani. Kinullázza a parancssort, majd kilép.
  - SHIFT+F5: a Copy_my meghívása, majd kilépés. Ez a helybenmásolás miatt kell.
  - Insert: az aktuális sor kijelölését megfordítja, majd egy sort lejjebb lép. Frissíti a selbart (CAct::RefreshSelBar()), majd kilép. Ügyelni kell a kiválasztott méret és szám helyes megállapítására!
  - CTRL+PGUP/PGDN: Egy szinttel feljebb lép illetve lejjebb. Itt közvetlen Enter_my() hívás történik, függetlenül a parancssor állapotától.
  - Space: csak NORMAL módban és könyvtáron nyomva megállapítja annak méretét a CAct::GetDirSize függvény hívásával.
- ChPanel(int): a panelek aktiváltságának megváltoztatása: aktívból passzív, passzívból aktív. A megadás lehet relatív vagy abszolút. Azaz hogy a kettő megcserélődjön, vagy esetlegesen direktbe mi mondhatjuk meg, hogy melyik legyen az aktív. Az act és a nact pointerek beállításaival kell trükközni. A végén a könyvtárpozíciót (dirloc) is színezni kell, és a parancssor melletti útvonalat is a helyesre beállítani.

- Ctrl_Enter_my: a parancssorhoz hozzáadja az aktuális file nevét. Ha a SHIFT is le van nyomva, akkor útvonallal együtt. Ha a 0. soron csináljuk, akkor csak az útvonalat adja hozzá.
- RereadConfig(): újraolvassa a konfigurációs file-okat. Ennek menete:
    1. Megpróbáljuk megnyitni a felhasználó home-könyvtárában levő „.vc/vc.cfg”-t, ha nem sikerült, akkor a programpath változóban meghatározott helyről próbáljuk megnyitni a „vc.cfg” file-t, siker esetén ezt átmásoljuk az első megnyitás helyére, majd megint megnyitjuk. Ha ezután sem sikerült megnyitni, akkor hibát dobunk és kilépünk.
    2. Sikeres konfigurációs file megnyitás esetén elkezdjük azt olvasni, nagyrész jelző sorok után kutatva. Ezek a következők lehetnek:
        - [general]: RereadConfig_gen függvény végzi a file további elemzését.
        - [color]: RereadConfig_col függvény végzi a file további elemzését.
        - [package]: RereadConfig_pac függvény végzi a file további elemzését.
        - [extensions]: RereadConfig_ext függvény végzi a file további elemzését.
    3. Beállítjuk a gyorsbillentyűket a ReadShortCuts()-szal.
    4. Beállítjuk a színeket a Refresh_col()-lal.
    5. Beállítjuk a könyvjelzőket a RereadBookmark()-kal és a RefreshBookmark()-kal.
- RereadConfig_gen: a konfigurációs file általános részének a beolvasásáért felel. A nem megjegyzés sorokat két részre bontja az '=' jel alapján. Az első rész határozza meg, hogy minek adunk értéket, a második pedig maga az új érték.
- RereadConfig_pac: hasonló mint az előző, de itt az első rész is tovább kell bontani a '_' alapján. Ez alapján határozzuk meg, hogy melyik tömörítésről van szó, és annak melyik tulajdonságáról. Vigyázunk arra, hogy egy tömörítés csak egyszer szerepeljen a listában (GetPacInd).
- RereadConfig_ext: ez teljesen a pac-hoz hasonló.
- RereadConfig_col: ez is teljesen hasonló mit az általános. De itt a színekre koncentrálunk. Nem közvetlenül az egyes elemek színét határozom meg külön, hanem sémákat, ezekhez a sémákhoz lesznek hozzárendelve az egyes elemek.
- GetPacInd(AnsiString): kezelőnév alapján határozza meg a kezelő helyét a pacs tömbben és annak indexével tér vissza.
- GetPacIndByExt: ugyanaz mint előbb, de itt kezelőnév helyett kiterjesztés alapján határozza meg az első előfordulását.

- GetExtInd, GetExtIndByExt: ugyanaz, mint pac esetben, de itt a exts tömbben keres.
- readcommandline(): a parancssor elemeit olvassa be a felhasználó home-jában levő „.vc/history” file-ból.
- writecommandline(): elmenti a parancssor és az ActionForm inputsarainak elemeit, de ehhez a beépített SaveToFile függvényt használja. (Vigyázat az ActionForm inputsorainak betöltését már maga az osztály végzi el, hisz nem tudjuk, hogy az mikor is jön létre pontosan.)
- readpositions(): a felhasználó előző ablakbeállításait tölti be, méretre és pozícióra vonatkoztatva. A beállításokat a felhasználo home-könyvtárában levő „.vc/positions” file-ból veszi, ha ez nincs, akkor az ablak középen jelenik meg (poDesktopCenter).
- writepositions(): elmenti az előbb tárgyalt értékeket: Top, Left, Width, Height sorrendben.
- ReadShortCuts(): a gyorsbillentyűket olvassa be a felhasználó home-könyvtárában levő „.vc/shortcuts” file-ból. Soronként olvas, addig amíg bír és addig ad sorban értéket a megfelelő elemeknek. Így megőrződik a csonka vagy régi file-okkal a kompatibilitás.
- Resize_my(): sok esetben lehet meghívni, nem csak ablakátméretezéskor. Az elemek pozícióját határozza meg az ablak mérete alapján. Nem túl bonyolult kiszámolni az egyes elemek egymáshoz viszonyított helyzetét, inkább macerás és sok munkával jár.
- ExecCmd: parancs elindítására szolgál bizonyos körülményekkel.
    1. fork() meghívása, a valami változóban tér vissza.
    2. valami -1 értéke esetén, a fork sikertelen volt.
    3. valami 0 értéke esetén az új processzről van szó. Ekkor megvizsgálásra kerül, hogy xshellben akartuk-e elindítani, ha nem, akkor simán indít, ha igen, akkor az execpath változóban beállított értékkel indít. Majd _exit(0)-val kilép.
    4. ha a valami nem 0, akkor ez az indítot processzt jelenti és a valamiben az indított/gyerek processz száma lesz. Ha a pwait be lett állítva az ExecCmd hívásakor, akkor megvárja annak befejeztét, ellenkező esetben berakja a processz számát a sprcss tömbbe.
- static AnsiSplit: egy meghatározott stringet egy minta alapján szétvág kettőre, úgy, hogy a minta nem fog előfordulni sem az elsőben sem a második részben sem. Rengeteg ellenőrzés kell, mert a stringek kezelése igen kényes. Ha a minta nincs benne a stringben, akkor azz első részben lesz az egész, a második pedig üres lesz.
- static AnsiCounter: egy stringben egy karakter előfordulásainak a számával tér vissza.

- HumanSize: egy nem „Directory” típusú szám olvasható változatával tér vissza, ha be van állítva a humanread bit. (K, M, G, T, P, E,Z, Y) A Tört számok végét levágja egy tizedesjegyig.
- static GetStringByMask: az egyszerűsitett maszkoknál jön elő. Csak annyit csinál, hogy a maszkban '*' előfordulásait lecseréli magára a stringre.
- static StrToChmod(AnsiString): a paraméterben megkapott jog string (pl.: rwxr-xr--) alapján kiszámolja a hozzá tartozó integert (a 0-ra rávagyoljuk a megfelelő biteket). Paraméternek elfogad 9, 10, 12 és 13 hosszú formátumokat is.
- RereadBookmark(): a felhasználó saját bookmarkját olvassa be, a „.vc/bookmark” file-ból. Először törli a mybookmark láncolt listát, majd a file-ból feltölti azt. A file minden egyes sora egy útvonalat tartalmaz.
- RefreshBookmark(): ez fogja a mybookmark alapján előállítani a tényleges és látható bookmark menüt. Először kitöröljük az összes elemet, hogy a mybookmarkból az összes változást fel tudjuk vinni. Majd egy elválasztót is hozzáadunk legvégül pedig az új bookmark-elem hozzáadásához szükséges menüpontot. Egyenlőre szerkesztés nincs!
- writebookmark(): a „.vc/bookmark” file-ba elmenti az éppen aktuális bookmarkokat, a mybookmark listán végigmenve.
- static MessageDlg_my: egy saját dialógus ablakot dob fel, melyen be lehet állítani a kérdést és a rá adandó válaszokat, maximálisan hat válasz lehetséges. Ez fogja megjeleníteni az InfoFormot.

3.3.2. A TInfoForm osztály

Ez az ablak egy kommunikációs ablak. A felhasználót egy bizonyos döntés elé állítja, és megengedi a döntés lehetőségét. Hasonló mint a beépített MessageDlg, de az nem enged elég szabadságot. Itt hat tetszőleges feliratú gombot helyezhetünk el. Ez a legtöbb esetben elég, nekem még sosem kellett több. Az ilyen típusú ablakokat teljes mértékben kívülről kell irányítani, azaz, mi legyen a gombok felirata, azok látszódjanak-e. Erre való a TForm1::MessageDlg_my függvénye melyben dinamikusan kezelhetjük a gombokat.
A form bármelyik gombjának a lenyomása a form bezárását vonja maga után. Ha entert ütünk az olyan, mintha az első gombot nyomtuk volna meg (ez a default értékű). Ha ESC-et ütünk, akkor az olyan, mintha a második gombot nyomtuk volna meg (ez a defaul cancel).
Ennek az osztálynak egyetlen egy változója van:

– ret: ez adja meg, hogy a dialógus ablak melyik gomb lenyomásával tért vissza. A konstruktorban a 2-es értéket kapja meg, ami a 2-es gombnak felel meg, ami ESC-re is meghívódik. Tehát a ret 2-es értéke alapból, a ne csinálj semmit.

3.3.3. A TConfigForm osztály

Ennek az osztálynak az ablakát a felhasználói dokumentációban eléggé részletesen leírtam, sőt aki a konfigurációs file-okról szóló részt is elolvasta, azoknak már az összes változó ismerős lehet. Ebben a fejezetben csak néhány fontosabb függvényre térek ki. Az egyik a beállítások mentése, a másik a gyorsbillentyűk kezelése.

A gyorsbillentyűk kezelése

Itt meg kell jegyeznünk, hogy a gyorsbillentyűk az egyes menüpontokhoz vannak eltárolva, ezek menüpontok eléggé statikusak, így kezelésük rendkívül körülményes lenne. Ennek a feloldására találtam ki, hogy kell csinálni egy tömböt olyan poiterekből, melyek ezekre a gyorsbillentyűkre mutatnak (nem érdemes az egész menüpontra mutatni, mert a többi részét nem használjuk). Ez a tömb lesz nekünk a „scs”, a tömb mérete pedig az „scn”. A RefreshSC () függvény fogja frissíteni a tömböt, hogy annak elemei jó helyre mutassanak. Emellett a tömb még eltárol egy magyarázó szöveget is, mely a beállításnál fog megjelenni, azért, hogy a felhasználó tudja mit is változtat meg. Miután létrehoztuk a tömböt beállítjuk a megfelelő értékeket. Erre sajnos nincs jobb módszer csak az, ha egyesével állítgatjuk be. Ez eleinte macerás lehet, de később már csak egy-egy új elem hozzávételével kell módosítani, ami nem kellemetlen (még akkor sem, ha az ember elfelejti módosítani, legfeljebb nem tudja majd megváltoztatni a felhasználó). A megváltoztatást a BGrid1 végzi, ami egy lista melynek elemeire kattintva az ember új értéket adhat meg. Delete billentyűvel pedig törölhet.

A beállítások mentése (SaveFile())

Ha röviden kellene jellemezni a mentést, akkor azt mondanám, hogy csak azok a dolgok kerülnek elmentésre, melyek eredetileg is benne voltak a konfigurációs file-ban (vc.cfg). Végig olvassa a file-t, és az egyes sorokat felbontja változóra és értékre, ő most ugyanezt a változót elmenti az új érékkel. Közben a régit elmenti „vc.cfg_old” néven. Ezt csak

manuálisan tudjuk visszaállítani. Ez is inkább macera a sok változó miatt, mint ördöngösség.

3.3.4. A TActionForm osztály

Ez az osztály felel azért, hogy a felhasználó jó helyre másoljon, vagy hogy meg tudja határozni mit keres, vagy hogy csatlakozni tudjon egy FTP-szerverhez. Tehát egy sokoldalú kommunikációs ablak ez. Két inputsora van (doon, doon2), melyek képesek információt elraktározni. Egyetlen nem látható változója a status melyben eltárolódik a felhasználó döntése: 0 – semmi, 1 – enter (azaz ok), 2 – mégsem (cancel).
Ez az osztály a programban úgy van használva, hogy az alkalmazás helyén mindig új lesz létrehozva, de ekkor honnan tudja meg szegény, hogy az inputsorainak mi volt a története. A folyamatos mentés és betöltés lassúvá teheti az egészet, ezért van az ActionForm globális példány ebből az osztályból. Ez létrejöttekor beolvassa történetet a „.vc/doon" és „.vc/doon2" fileokból, ez csak egyszer történik meg a program indulásakor valamikor. A mentést már a főprogram végzi kilépéskor. Az alkalmi ablakok innen fogják átmásolni a történetet és azokból való kilépéskor pedig visszamásolják azokat az új értékekkel. A billentyűlenyomásokat itt is egy „globális" függvény kezeli. Itt is érvényes a CTRL+LE/FEL billentyűkombináció a történetek elérésére. ESC természetesen a mégsemnek fele meg, az enter pedig az oknak.

3.3.5. A TViewForm osztály, azaz a nézőke

Ez az osztály felel a file-ok tartalmának megjelenítéséért. Sok-sok gond volt vele, amíg ez így kialakult, ahogy most van. Még mindig sok vele a gond, de már kezd alakulni. Eleinte a megjelenítést egy belső komponens végezte, mely a html oldalakat még szépen meg is jelenítette. Ennek több hátránya volt: nem volt jól kezelhető, azaz nem tudtam billentyűzeti szempontból jól kezelni, a nagy file-okkal nem boldogult megfelelően, keresést sem lehetett rá jól implementálni (sőt sehogyan). Egyik komponens sem volt képes nagyon nagy file-okat kezelni rendesen, ezért eldöntöttem, hogy kézbe veszem a dolgokat. Fogtam a legegyszerűbb komponenst, amire írni lehet és úgy kezelem, mint egy karakteres képernyőt, csinálok hozzá saját scrollbart és csak annyit jelenítek meg amennyi éppen elfér, így nem kell az egész file-t a memóriában tartani, hanem elég annak egy kis töredékét. Hátrány, hogy scrollozáskor mindig olvasni kell a file-ban, de a tapasztalatok szerint ez nem ront semmit a hatásfokon,

így ez a megoldás jónak bizonyult. Ekkor viszont az egész file olvasását saját magamnak kellett megvalósítani. A módszer látjuk majd később. Most nézzük meg melyik változó mire jó:

- filesize: a file mérete lesz eltárolva benne. Fontos, mert a file végére csak ezzel az adattal tudunk pozicionálni (no ez így nem igaz, de én így teszem).
- filename: a file neve útvonallal, hogy meg lehessen nyitni (külsőleg megváltoztatható).
- lwidth: ennél hosszabb sorok automatikusan törésre kerülnek, mert így nem lehetséges egy egy soros file-t teljesen a memóriába olvasni.
- lheight: ennyi sor látszódik egyszerre a nézőkében.
- line_num: technikai változó, olvasásnál használt a sorok számolására.
- herestream: ez adja meg azt, hogy a fileban éppen hol vagyunk, azaz a nézőke bal felső sarkában levő karakter pozíciója a file-ban (az egyik legfontosabb változó).
- hereline: az éppen aktuális sor száma, nem használt, mert kezelése lehetetlen. Például ha end-et nyomunk, akkor nem lehet végigkeresni az egész file-t, hogy megszámoljuk, hány sor is tartalmaz.
- herelinememo: ez azt határozza meg, hogy az egyszerre beolvasott sorok közül hol tartunk. Az eleső és az alsó harmad elérésével új részt kell beolvasni.
- viewer_lines: az egyszerre beolvasott sorok.
- findstring: a keresés stringje.
- findfromhere: a keresés kezdeti pozíciója. Új kereséskor ez a herestream-mel lesz megegyező.
- free: a nézőke szabad-e.

Most nézzük a legfontosabb függvényeket. Annak ellenére, hogy karaktereket olvasunk, az egyik legfontosabb egység a sor lesz. Egyszerre a memóriában mindig a látható soroknak a háromszorosa van, melyek legfeljebb lwidth hosszúak lehetnek, így a maximálisan lefoglalt memória töredéke a tényleges file méretének nagyon nagy file-ok esetén, természetesen kisebb file-oknál nem szükséges még ekkora sem. A legfontosabb függvény az i. pozíción történő olvasás:

- read(int): a paraméterben megadott helyen olvas háromszor ablaknyit. A pozíciótól előrefelé és hátrafelé is történik olvasás. Előrefelé egy hátrafelé két ablaknyi. Így a külső keretben szabadon mozoghatunk újbóli olvasás nélkül (cache-szerűség).
- refresh(): a beolvasottakat szükséges megjeleníteni is, nos ez a függvény a memóriából a kijelzőre dobja a herelinememo-tól a sorokat lheight hosszon. Fontos megjegyezni, hogy

a frissítés idejére a Viewer_my->Lines Update-jét érvényesítjük, mert ilyenkor nem jelentkezik vibrálás.

- up(int=1), down(int=1): a fel- illetve le kurzorbillentyűk használatakor a látható részt illeszti lejjebb illetve feljebb. Ha a cache legtetejére értünk up során, akkor újraolvasás történik. Down-nál az alsó harmadnál történik az újraolvasás, mivel ha ekkor mennénk túl, már nem tudnánk honnan olvasni.
- FindInFile_my, FindInFile: a FindInFile_my felel a kérdező ablak feldobásáért és a keresés első indításáért. Itt adja meg a felhasználó a maszkot melyre keres. Nem kell '*'-ot írnia a maszk elejére és végére, mert a program ezt automatikusan megteszi. A keresést a FindInFile végzi, ami teljesen hasonló a CAct-nál tárgyalttal, de itt még ügyelni kell, arra hogy az ablakot oda kell pozicionálni és a talált szöveget ki kell jelölni (highlight).

3.3.6. A TMultiRenameTool osztály

Ez az osztály végzi el az átnevezésnél szereplő file nevek átnevezését egy formátum alapján.
A felhasznált válzozók:

- num: ennyi file-t kell majd átnevezni.
- status: ugyanaz, mint TActionForm esetén.
- onames: a régi file-nevek.
- nnames: az új filenevek.
- NameFormat: ide írhatja a felhasználó a kívánt formátumot, melynél minden karakterleütés után újra számoljuk az új neveket.

A felhasznált függvények:

- static GetNewName: ez a függvény kezdi meg az elemzését a formátumnak. A változókat illetve a felhasználható függvényeket '%' jellel kezdjük. Ha ez egy ilyen jelet talál, akkor egyből meghívja az interpretert. Tulajdonképpen ez a függvény végzi el az átalakítást a régi névről az újra, a formátum szerint. Ha nem '%'-t olvas, akkor az olvasott karaktert belerakja az új névbe is.
- Interpret: a lényegi részt ez végzi, azaz ha valaki talál egy függvényt (ami '%'-kal kezdődik), akkor ezt a függvényt hívja meg, még ez is (azaz rekurzívan). Olvassa folyamatosan a karaktereket és egy LL(1)-es elemzőhöz méltóan az első illeszkedés (felismert függvénynév) esetén már tudja is, hogy mit kell tennie. Egyes függvények paramétereket igényelnek, ezt az interpreter tudja és beolvassa. Hiba esetén nem történik

semmi, azaz nincs hibaüzenet, esetleg az eredmény nem vagy hibásan jelenik meg. Ha egy függvény szám paramétert igényel, akkor '%' után közvetlenül a számot is megadhatjuk. A '%'-el kezdődő kifejezések függvényhez méltóan visszatérési értékkel rendelkeznek és tetszőlegesen egymásba ágyazhatóak.

- mp3id: az egyik formátum-függvény lehetővé teszi, hogy az új névbe mp3 file-ok id3v1 tagjeiből információkat illesszünk be.

Ezzel nagyjából végigtárgyaltuk az összes osztály. A dokumentációt érdemes frissíteni, mert idővel bővülhet.