Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Gracia Dániel

SNIPPET ALKALMAZÁSA RPG PROGRAMOZÁSBAN

EGYETEMI KONZULENS

Dr. Renczes Balázs

KÜLSŐ KONZULENS

Kern András
Unicredit Services S.C.p.A

Budapest, 2019

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Gracia Dániel**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2019. 12. 12.

...…….
Gracia Dániel

Összefoglaló

Szakdolgozatom során a feladatom egy általános karbantartó alkalmazás létrehozása volt RPG nyelven, illetve ehhez az alkalmazáshoz kapcsolódóan egy snippet könyvtár létrehozása az IBM Rational Developer for i fejlesztőkörnyezetében. A snippet az RDi-ban megjelent újítás, egy előre definiált kódrészlet, amely lehetővé teszi a programozás során a program moduláris felépítését.

A dolgozatomban bemutatásra kerül az IBM AS400-as rendszere, ugyanis ezen a rendszeren történik az RPG nyelvű programfejlesztés az Unicredit Services-nél. Emellett bemutatom a rendszer fejlődését a megjelenésétől kezdve, illetve üzelti státuszát napjainkban.

Fő célom a fejlesztőcsapat produktivitásának növelése volt a Rational Developer for i, illetve a snippetek bevezetésével. Első lépésként létrehoztam egy általános karbantartó alkalmazást, amely az Unicredit Services-nél gyakran előforduló alkalmazás típus. Ehhez az alkalmazáshoz szükség volt egy display fájlra, amit DDS-sel (Data Description Specification) írtam le, egy fizikai fájlra, ami a megjelenítendő adatokat tartalmazta, illetve egy RPG programra, ami megvalósította a szükséges funkciókat, mint pl. új rekord létrehozása, módosítása, illetve keresés. Mivel az alkalmazás kellően általános volt, így lehetőségem nyílt egy snippet könyvtár létrehozására, amely a későbbi karbantartó alkalmazások létrehozásában és karbantartásában nyújt segítséget. Feladatom volt a snippetek összegyűjtése, illetve rendszerezése is, az utóbbit a specifikációnkénti (F, D, C, stb.) megjelenítéssel oldottam meg. Egy jól megírt és struktúrált snippet könyvtár konzisztenciát biztosít a fejlesztőcsapatban és lehetővé teszi a moduláris alkalmazásfejlesztést, ennek első lépése volt az általam létrehozott snippet könyvtár.

Abstract

During my thesis my task was to create a general maintenance application written in RPG, besides that, the other part of the task was to create a snippet library in the IBM Rational Developer for i integrated development environment. Snippet is a new feature of RDi, that is a predefined chunk of code, which enables the programmer to build a modularized application.

In my thesis, I give an overview of the AS400 system, since the development of applications in RPG is done on this system at Unicredit Services. Besides that, I present the development of this system from its appearance and its business status nowadays.

My main goal was to increase the productivity of the developer team through the introduction of Rational Developer of i and its snippets. The first step was to create a general maintenance application, that is commonly used at Unicredit Services. In order to create this application I needed a display file, which I coded in DDS (Data Description Specification), a physical file, which contains the used data, and last the RPG program, which realizes the functions of the application, like creating a new record, changing an existing one or searching for one. Since this application was general enough, I could implement a snippet library, which will help programmers to develop and maintain applications. My task included collecting and organizing snippets, the latter was done by specifications (F,D,C etc.). A well written snippet library provides consistency among the developers and makes modularized application development possible, and the first step to this is a snippet library created by myself.

1 Bevezetés

Manapság a banki informatikában az egyik legelterjedtebb programozási nyelv az RPG (Report Program Generator), amelyet az IBM fejlesztett 1959-ben. Annak ellenére, hogy egy viszonylag régi programozási nyelvről van szó, a mai napig széleskörben használják pénzügyi és szolgáltató rendszerek szoftveres támogatására.

Az RPG-t eredetileg lyukkártyás gépekre fejlesztették, később terminálképernyőn elérhető szoftveres kódszerkesztőt kapott, ma már viszont elérhető az IBM által fejlesztett Eclipse alapú grafikus fejlesztőkörnyezet, az IBM Rational Developer for i. Annak oka, hogy a mai napig elérhető és széleskörűen használt ez a programozási nyelv többek közt a nyelv folyamatos fejlesztésének köszönhető. A fő célja az RPG-nek, ahogy a neve is utal rá, riportok generálása adat fájlból, de a folyamatos fejlesztésnek köszönhetően ma már komplexebb műveletek elvégzésére is alkalmas. A nyelvet eredetileg az IBM 1401-es gépre fejlesztették, majd később bevezették az RPG III-at az AS/400-as (mai nevén IBM System i) rendszerre, amely szintén az IBM termékcsaládhoz tartozik.

Általánosságban elmondható, hogy a projekt alapú szoftverfejlesztés során fontos, hogy az előírt határidőre meglegyen a specifikációknak megfelelő alkalmazás, emiatt a cégek igyekeznek a produktivitás folyamatos növelésére az egyre fejlődő technológia eszközök segítségével. Munkám során eleinte a már rég bevált zöld képernyős rendszeren dolgoztam, amely az AS400-as rendszeren fut, viszont feladatom az nemrég bevezetett fejlesztő környezet, a Rational Developer for i használatára is kiterjedt. A cél az RD által támogatott eszközök segítségével a fejlesztés gyorsaságának növelése, külön figyelmet fordítva a snippetek használatára, amelyek nagyban tudják segíteni a fejlesztők munkáját.

Feladatom első része egy általános, az Unicredit Servicesnél használt karbantartó alkalmazás megírás volt, eredetileg zöld képernyőn, később a Rational Developer nyújtotta eszközök használatával, mint pl. a snippet.

Továbbá feladatom volt annak meghatározása, hogy a fejlesztés során mely részeket érdemes és lehet megvalósítani snippetekkel, illetve ezeket a kódrészleteket összegyűjtve egy snippet könyvtár létrehozása, amely a továbbiakban támogatja a fejlesztők munkáját és konzisztenciát biztosít a csapaton belül. Fontos azonban megjegyezni, hogy a Rational Developerre való áttérés nem jár azonnal a produktivitás növekedésével, ugyanis egy tapasztalt, több, mint 10 éve zöld képernyőn dolgozó fejlesztő esetében az új funkciók és eszközök használata eleinte lassabb, mint a már megszokott rendszeren. Az viszont egyértelmű, hogy a jövőben az RPG nyelvű alkalmazásfejlesztésre legalkalmasabb fejlesztőkörnyezet a Rational Developer for i.

2 AS400-as számítógépcsalád bemutatása

Az AS/400(Application System 400, vagy másnéven az IBM System i egy középkategóriás (midrange) számítógép a mikroszámítógép és az ún. „mainframe” számítógép osztály közé esik, az IBM vezette be 1988-ban. Már a megjelenésétől fogva mérföldkőnek számított a megbízhatóság és az egyszerű használatának köszönhetően. Az egyik tulajdonsága, ami hozzájárult a termék hosszú idejű fennmaradásához a magas szintű utasítás készlete (high level instruction set), vagyis a TIMI (Technology Independent Machine Interface). A TIMI virtuális utasításkészlete független a CPU utasítás készletétől, ennek következtében a felhasználói programok a TIMI utasításai mellett a CPU utasításaiból is állnak. A TIMI utasítások egy külön lépesben fordítódnak le a processzor utasítás készletére, amely a fordítás utolsó lépése. A CPU eleinte CISC volt, ezt később lecserélték az IBM RS64 néven ismert RISC CPU-ra, illetve míg ’95-ben az órajel sebesség 55/75 MHz volt, a legújabb 2017-es gépnél ugyanez már 4GHz volt, ezzel is igazodva a minél gyorsabb működés igényéhez. A RISC típus előnye a CISC-cel szemben, hogy az ott utasítások hossza megegyezik, míg a CISC esetén az utasítások hossza változó, illetve általánosságban elmondható, hogy adott utasításhosszra tervezett számítógépek sokkal hatékonyabbak.

2.1 Az AS400 fejlődése

Az IBM miniszámítógépeinek („minicomputer”, ma már „midrange” néven fut) fejlődése a System/3 miniszámítógép rendszer bejelentésével kezdődött 1969-ben. Ez volt az első számítógép, amit az IBM fejlesztett egyedül, és amely lyukkártyákra volt tervezve. A System/3 jelentős technológiai fejlődést mutatott abban az időben, és ezt követő számítógépek ugyanezt az eredetileg MIT által kifejlesztett hardware design-t követték.

A System/32 volt a System/3 utódja. 1975-ben mutatták be és lyukkártya alapú volt, viszont már lehetőség volt billentyűzetről bevinni adatot, illetve képernyő is tartozott hozzá. Eleinte a számítógép egyfelhasználós volt, később az IBM létrehozott egy új modellt, amely egyidejűleg öt adatbeviteli terminált támogatott.

Ezt követően 1980-ban mutatták be a System/38-at, amely több innovációt tartalmazott. Ezek közül egyik a nagyszámú beépített funkció, mint például a relációs adatbázis – amely először itt jelent meg – magasabb szintű biztonsági rendszer, széleskörű backup/recovery eszközök, illetve egyéb funkciók. Továbbá egyedi volt a System/38 a fejlett operációs rendszer, illetve az objektum alapú (objektum orientált rendszerhez hasonló) architektúra tekintetében.

1988 júliusában megjelent az AS400, amely eleinte nem tűnt radikálisan jobbnak a System/38-hoz képest, viszont az évek során a számítási kapacitás évente átlagosan 10%-kal növekedett az upgrade-nek köszönhetően (anélkül, hogy a gépet lecserélnék). Ez a gyors fejlődés eleget tett a business követelményeknek. Mindemellett az AS400 egyik kulcsfontosságú tulajdonsága a hardwarefüggetlensége, így nem kellett a migráció problémájával foglalkozni, csupán a disk és a memória kapacitás növelésével megkapták az új verziót. Nem kellett lemásolni és újra fordítani a programokat, illetve nagy adatbázisokat újra tölteni az új gépen, a legtöbb esetben a migrációt pár nap alatt el lehetett végezni. A folyamatos fejlesztésnek köszönhetően a mai AS400 modellek elérhetőek már 700$-tól elérhetőek, egészen 2M$-ig, ezzel a szinte bármely nagyságú cég a saját igényeinek megfelelő modellt választhatja.

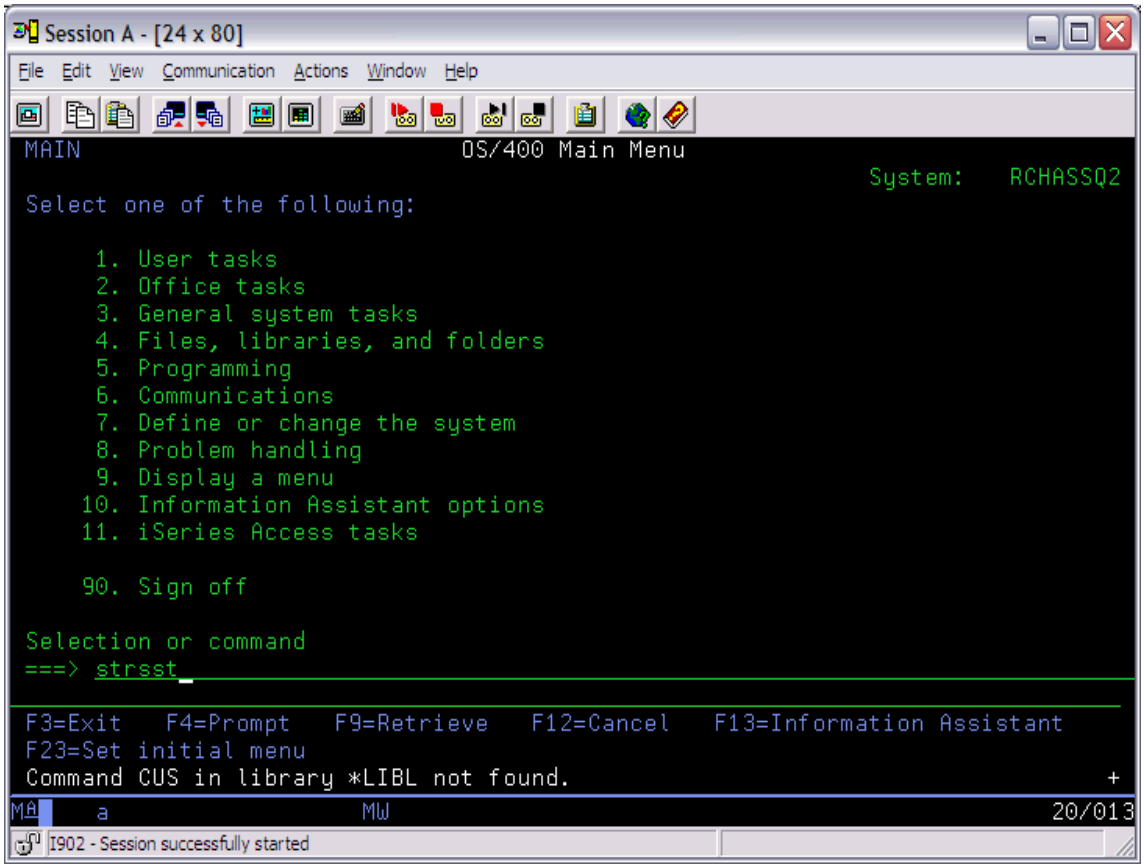
Természetesen a következő években is tovább fejlődött a termékcsalád az egyre növekvő igényekhez igazodva, 1997-ben bemutatták az e-Series-t (Enhanced Series), amelyek akár 12 processzort is tartalmaztak.

Az AS/400 az RPG mellett számos programozási nyelvet támogat, mint például a C, C++, Java, Perl és a Phyton. Annak ellenére, hogy magát a számítógépet több, mint 30 éve fejlesztették ki, a mai napig számos helyen használják, mivel rendkívül biztonságos, és emiatt használják kórházakban, bankokban és kormányzati szerveknél is.

2.2 OS400

Az AS400-on futó operációs rendszer az OS400, mai nevén i5/OS vagy IBM i, amely egyike az IBM Power Systems-en futó operációs rendszerek egyikének az AIX és a Linux mellett. A rendszer egyik sajátossága, hogy relatíve kevés támogatást igényel a normál működése során, a TCO-ja (Total Cost of Ownership) sokkal alacsonyabb, mint a versenytársainak. Mindemellett ez volt az egyik első az objektum alapú rendszerek

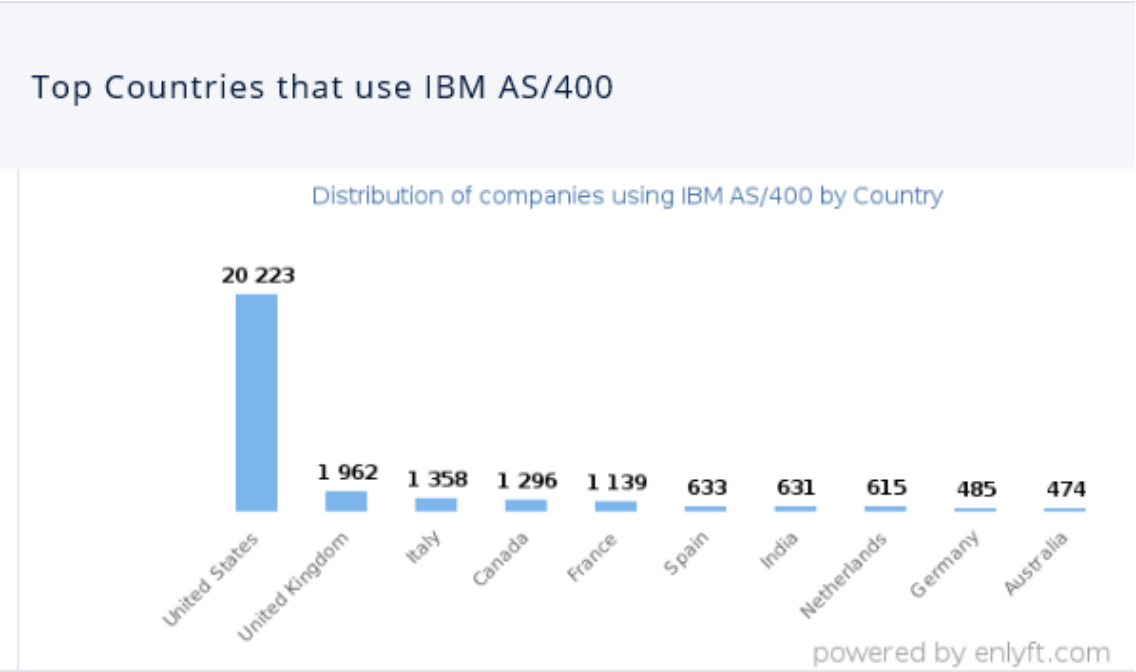
között, vagyis itt nincsenek fájlok, csak különböző típusú objektumok. A rendszer másik fontos sajátossága, hogy tartalmaz egy integrált adatbázis-kezelő rendszert, az IBM DB2-t, illetve támogatja az SQL-t is. AZ AS400-hoz hasonlóan az OS400-at is folyamatos fejlesztik, a jelenlegi legfrissebb verziót (7.4) 2019 áprilisában adták ki.



2.1 ábra: OS400-as operációs rendszer

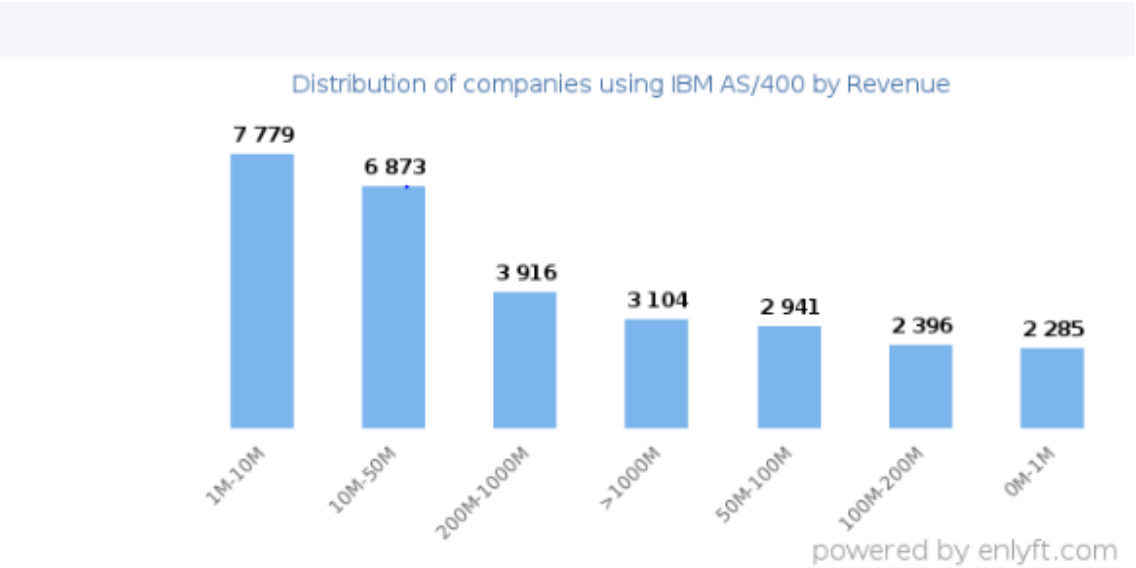
2. 3 Üzleti státusza napjainkban

Számos Bank és Biztosító használja a mai napig az AS400-as gépeket, többek között a Barcleys Bank Angliában, Banco de Santander Spanyolországban - ahol emellett RPG nyelven történik a fejlesztés - illetve a Banque BNP Paribas Franciaországban. Az európai jelenlét mellett a cégek nagyrésze az USA-ban található.



2.2 ábra: AS/400-at használó országok eloszlása, Forrás: [3]

Egy 2015-ös adat szerint egyedül az USA-ban több, mint 20000 AS/400-as gépet használnak (lásd 2.2 ábra), és számos európai országban is 1000 fölött van a számuk, ezzel igazolva, hogy a mai napig fontos szerepet tölt be az üzleti életben. Mindemellett az AS400 33%-os részesedéssel piacvezető a Hardver Szerver kategóriában, ezzel lehagyva a HP-t (16,9%), a Unisys-t (5.4%) és a Dell-t (5.4%) is.



2.3 ábra: AS/400-at használó cégek eloszlása méretük szerint, Forrás: [3]

További érdekesség, hogy a cég bevételének nagyságától függetlenül mindenhol jelen van, kezdve a kisebb, maximum 1M$-os bevételű cégektől az óriáscégekig, amelyek bevétele az 1000M$-t is meghaladja (lásd 2.3 ábra).

Az AS400 (mai nevén már i Series) mai napig tartó jelenléte nagyrészt annak köszönhető, hogy leváltása rendkívül költséges és nehéz, számos precedens van arra, hogy egy másik platformra való áttérés (pl. Unisys Mainframe) után mégis visszatértek az i Series-hez új rendszeren felmerült nehézségek miatt.

Az AS400-ra írt alkalmazások nagy része RPG nyelven van (kb. 70%), viszont előfordulnak COBOL, illetve C/C++ nyelven írt programok is. A COBOL (common business oriented language) az RPG-hez hasonló programozási nyelv, amelyet főként üzleti, pénzügyi és adminisztratív rendszerekben használnak. A nyelvet 1959-ben fejlesztették ki, a ma már viszonylag elavultnak számító nyelvet viszont még mindig használják mainframe típusú számítógépeken és az RPG mellett az egyik legbiztonságosabb nyelvként tartják számon, emiatt is van még használatban biztonság kritikus környezetben (pl. bank). Az RPG-hez képest hasonló hatékonyságú nyelv, viszont az évek során nem ment keresztül olyan mértékű fejlődésen, mint ami az RPG-nél tapasztalható volt. Mindazonáltal a csökkenő népszerűsége és a COBOL fejlesztők számának csökkenése miatt folyamatosan történik a programok migrációja más platformokra és más nyelvekre történő átírása, így manapság a COBOL programozás célja főként a már létező alkalmazások fenntartása.

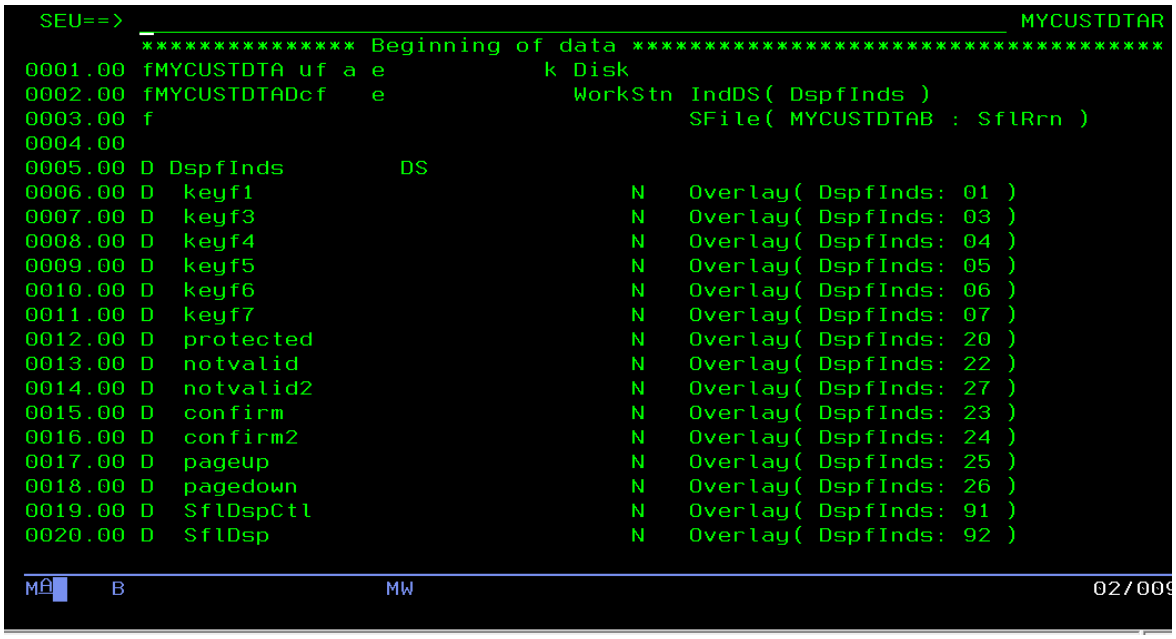
2.4 RPG

Mivel maga a nyelv nem annyira közismert, mint pl. C++ vagy a Java, így röviden bemutatom a tulajdonságait. Az RPG nyelvnek az egyik fő sajátossága, hogy a program 6 funkcionálisan elkülöníthető részből („specifications”) tevődik össze, amelyeket meghatározott sorrendben kell bevinni. A sorrend a következő: Control Specifications, File Description Specifications, Definition Specifications, Input Specifications, Calculation Specifications és Output Specifications. Mindegyik specifikáció külön feladatot lát el.

- Control Specifications: a fordítónak szolgáltat információt a programok gerenálásával és futtatásával kapcsolatban

- File Description Specification: Felsorolja az összes fájlt, amit a program használ
- Definition Specification: Felsorolja az összes adatot (változók, adatstruktúrák stb.), amit a program használ
- Calculation Specification: Felsorolja az adatokon végzett műveleteket
- Output Specification: Megadja a kimeneti rekordokat és mezőket, amit a program használ

A specifikációk közül mindegyik opcionális, viszont legalább egynek jelen kell lennie a programban. A függőleges tagoltság mellett a programozás során figyelni kell a bevitt kód pozícionálására, ugyanis az egyes utasítások, változók, kulcsszavak csak meghatározott pozícióban (oszlopban) érvényesek.



2.4 ábra: Zöld képernyős szerkesztő felület

A további fejlesztéseknek köszönhetően már létezik a RPG-nek egy free form verziója, amely lehetőséget ad a többi modern programozási nyelvhez hasonló használtra, ugyanis nem kell meghatározott oszlopba írni a kódot (8-80 oszlop között). Ezáltal a free formban írt kód sokkal könnyebben olvasható, illetve a változók is beszédesebbek, ugyanis nincs korlátozás a hosszra.

Az AS400-hoz hasonlóan az RPG is folyamatos fejlődésen ment át. Az RPG III-as verzióban jelent meg először az IF-ENDIF blokk, a DO ciklus és a szubrutinok bevezetése is ehhez a verzióhoz köthető. 1994-ben jelent meg az RPG IV, ami további

újításokat hozott a fejlesztők számára, mint pl. statikus és dinamikus linkek, eszközök a C-ben megírt rutinok hívására. 2001-ben volt az egyik leghasznosabb és a fejlesztők munkáját legjobban felgyorsító frissítés a free format bevezetése. Mindazonáltal a kód szerkesztésére még mindig gyakran a SEU-t (Source Entry Utility) használják, amely egy egyszerű zöldképernyős szerkesztő syntax ellenőrzést támogató funkció nélkül, viszont itt is történt előrehaladás, mint pl. a Rational Developer megjelenése.

2.5 Szoftver kiválasztása

Az RPG nyelvű alkalmazásfejlesztéshez használt szoftver megválasztásánál több lehetőség is adódott. A választásnál az alábbi szempontokat vettem figyelembe:

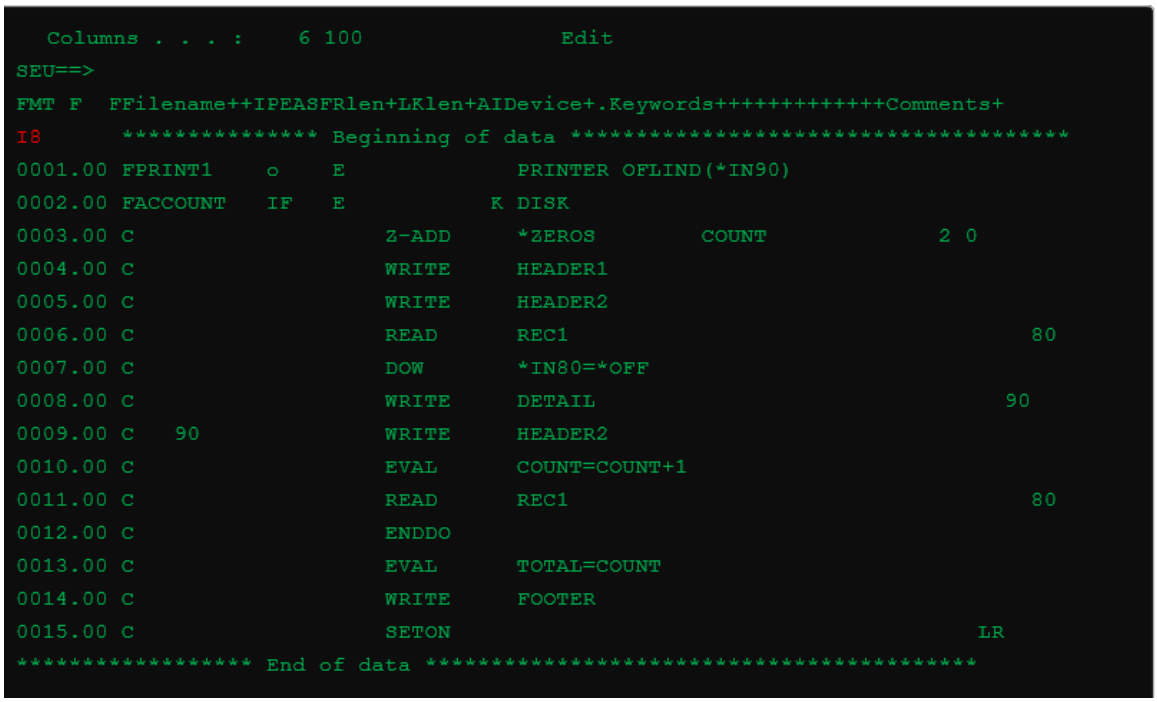
- Legyen olcsó vagy legyen belőle próba verzió
- Tartalmazzon olyan eszközöket, amelyek segítségével növelhető a fejlesztőcsapat produktivitása
- Jelentős fejlődést mutasson a jelenleg használt rendszerhez (SEU/PDM) képest

Ezeket a szempontokat figyelembe véve az alábbi szoftverek merültek fel:

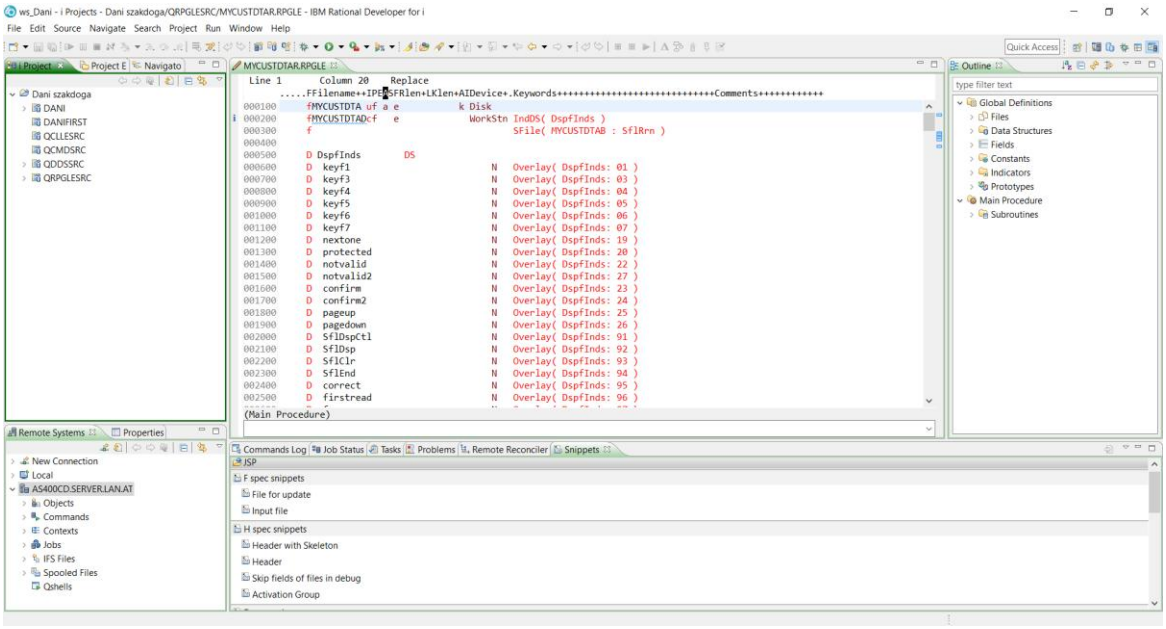
- Orion: Tartalmaz syntax highlighting-ot RPG, COBOL, DDS és CL nyelvekre. Egy viszonylag egyszerű ingyenes kódszerkesztő.
- MiWorkPlace: A program éves díja 50$, de elérhető próba verzió is. A program segítségével navigálni lehet az IBM i könyvtárban, támogatja a source fájlok és spool fájlok fejlesztését. A Rational Developerhez hasonlóan Eclipse alapú, viszont nem tartalmaz annyi eszközt.
- Rational Developer for i: A legjobb választás egyértelműen a Rational Developer for i: Számos olyan funkciót tartalmaz, amelyet a többi szoftver nem, ezeket később részletesen tárgyalom. Az egyetlen negatívum a magas ára, az éves díja felhasználónként 940$, emiatt sok cég nem veszi meg a licenszet és inkább más, olcsóbb alternatívát használ. Mindazonáltal jó ár/érték arányt képvisel, esetemben elérhető volt egy 60 napos próbaverzió, így emellett döntöttem.

2.1.1 Rational Developer

A Rational Developer for i egy Eclipse alapú integrált fejlesztőkörnyezet (IDE) alkalmazások létrehozásához, fenntartásához, illetve modernizálásához az IBM i platformra. Számos fejlesztői eszközt tartalmaz, mint pl. szerkesztő, analizáló, illetve a debugger a széleskörűen használt Eclipse frameworkben így gyorsabb és egyszerűbb alkalmazásfejlesztést tesz lehetővé. A 2.5-ös és 2.6-os ábra illusztrálja az alkalmazásfejlesztés platformjának két alternatíváját, a jelenleg használt zöldképernyős PDM-et (Project Developement Manager), illetve a Rational Developert.



2.5 ábra: PDM



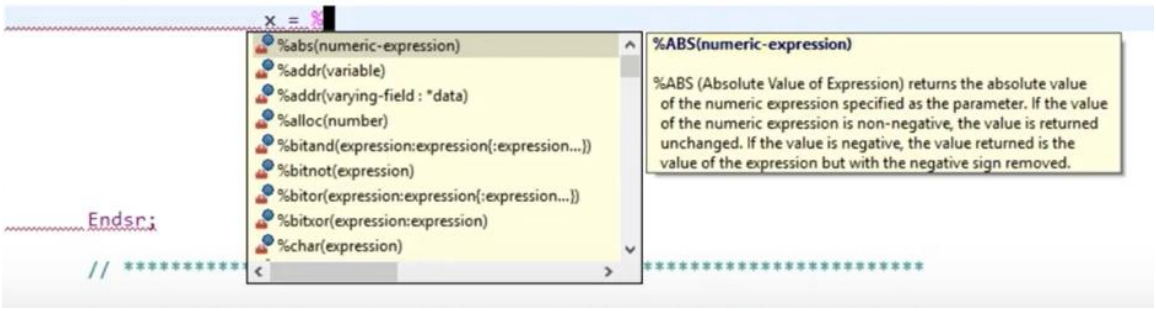
2.6 ábra: Rational Developer for i

A Rational Developer a mai elvárásoknak megfelelő modern fejlesztőkörnyezet. Az egyik szembetűnő különbség a két szerkesztő között a kulcsszavak és utasítások külön színnel való kiemelése, amely nagyban javítja az átláthatóságot. Az utasítások gépelése közben elérhető a Content Assistant, ami egy felugró ablak a lehetséges utasítások listájával, illetve az adott utasítás leírásával. Ez a funkció is nagymértékben segíti a fejlesztő munkáját, ugyanis nem kell külön az utasítások pontos nevét és szintaxisát megkeresni a több száz oldalas RPG Reference Guide-ból.

A kódszerkesztőben a fixed formban írt kódot egyszerűen át lehet konvertálni free formba, viszont ez csak a C specifikációban írt kódra érvényes. A többi specifikáció átkonvertálására elérhetőek eszközök az interneten (pl. Craig Lutredge's free tools, forrás: [8]).

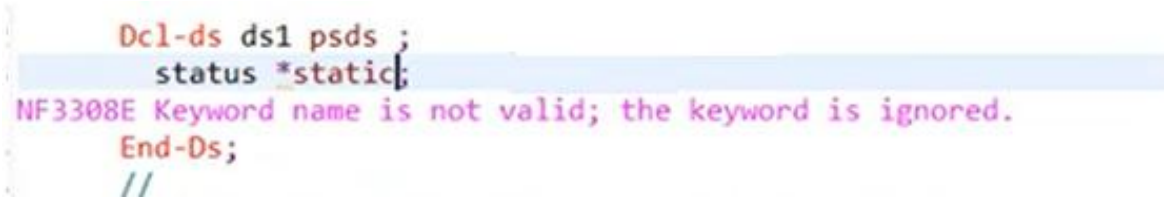
Az Outline menüben (ami nincs a PDM-ben) kategorizálva megjelennek a kódban lévő külső és belső erőforrások (konstansok, indikátorok, prototípusok), így sokkal könnyebb az ezek közötti navigálás.

A fájlok közötti navigálást nagyban meggyorsítja az RDi i projects navigátora, ez is jelentős előrelépés a PDM-hez képest. A PDM 80*24-es felbontása is korlátozta a fejlesztőt, ez korlátozó tényező a Rational Developernél megszűnt.



2.7 ábra: Content Assitant

Míg a SEU esetében az esetleges elírások és a rosszul használt kulcsszavak okozta hibát csak fordítás után jelezte a rendszer, addig a Rational Developernél a Syntax Checker ezt már a kód írása közben elvégzi és az adott sornál jelzi.



2.8 ábra: Syntax Checker

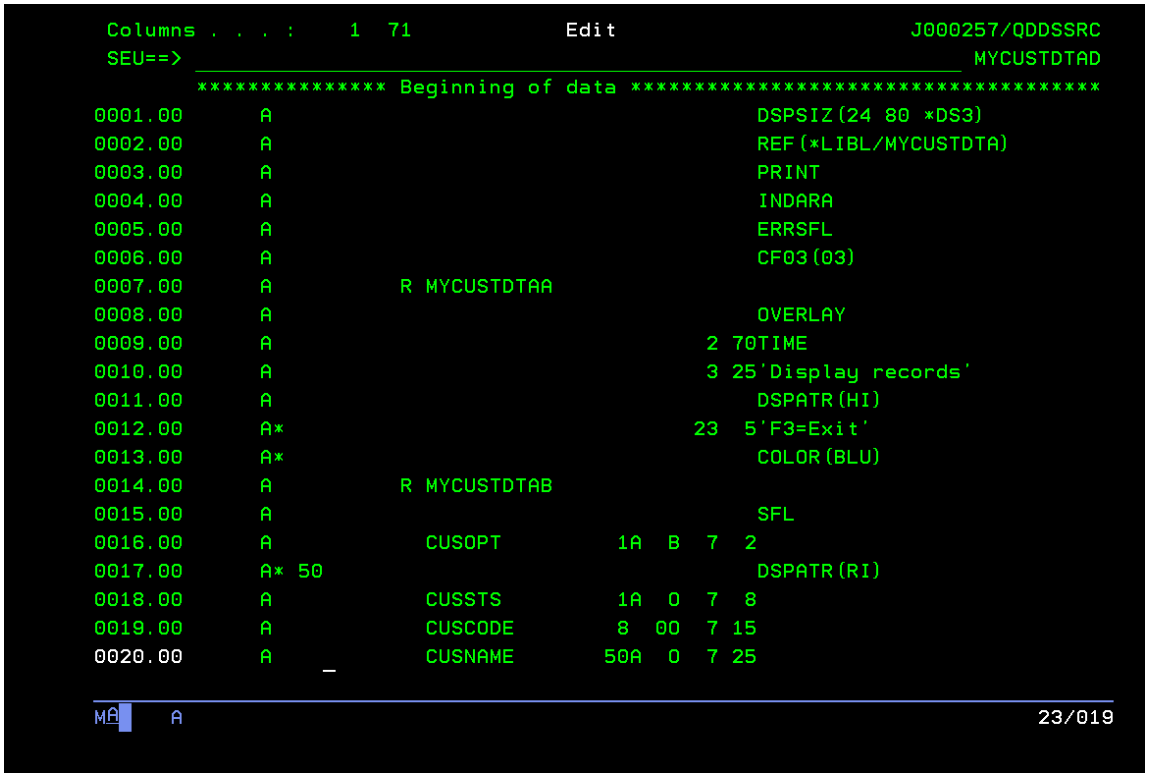
A template használat lehetősége is a Rational Developer egyik újítása. A fejlesztőkörnyezet ugyanis számos előre definiált template-t (sablont) tartalmaz, amelyek nagyon hasznosak lehetnek pl. SQL utasítások beszúrásakor és sok egyéb helyzetben. Emellett a felhasználó is definiálhat templateket a saját igényeinek megfelelően. Az RDi-nak a templatehez hasonló másik eszközét, a snippeteket későbbi fejezetekben tárgyalom. Természetesen elérhető még számos funkció, ebben a fejezetben csak a legfontosabbakat tárgyaltam.

3 Az RPG alkalmazás főbb elemei

3.1 Display file

Ahhoz, hogy egy működő képernyős alkalmazást hozzunk létre, az RPG fájl mellett szükség van egy úgynevezett display file-ra, amelyet DDS (Data Definition Specification) segítségével definiálhatunk. A display file határozza meg tulajdonképpen a képernyőt, amin a program fut, és amelyen keresztül a felhasználó kommunikálhat a programmal.

Maga a fájl rekordokból áll, amelyek a képernyő különböző részeit reprezentálják (fejléc, menü, funkciósor) és ezekhez, különböző funkciókat társíthatunk.



3.1 ábra: Display fájl zöld képernyőn

Minden rekordnál lehetőség van az alábbiak meghatározására:

- A képernyőn a mező helye (pl a 7. sor 8.oszlopban kezdődik)
- A mező hossza
- Az adat típusa (karakter, zoned decimal, floating point)
- Mező típusa (input, output, both)

A pozíciókra, ahogy az RPG-nél, úgy itt is figyelni kell, pl. a 25.oszlopban R betűvel lehet jelezni, hogy egy rekordról van szó. A rekordon belül jelezni lehet különböző kulcsszavakkal a betűk színét (COLOR(BLU) - kék), védetté lehet tenni (DSPATR(PR) – protected), illetve aktiválni lehet az egyes funkció billentyűket (CF03(03) – F3 aktiválása).

Egy további érdekes sajátossága a DDS-nek, hogy meg lehet adni indikátorokat az egyes sorokban, amelyeket majd az RPG programban beállítva lehet a képernyő tartalmát, illetve tulajdonságait aktiválni. Példaképpen felmerülhet, hogy egy adott programban a felhasználó képes egy rekordot megjeleníteni, illetve módosítani és ennek megfelelően a rekord védettségéhez (DSPATR (PR) – protected) tartozó indikátor ki és bekapcsolásával tudja a programozó letiltani, illetve engedélyezni az adott rekord szerkesztését a felhasználónak.

3.2 Fizikai és Logikai fájl

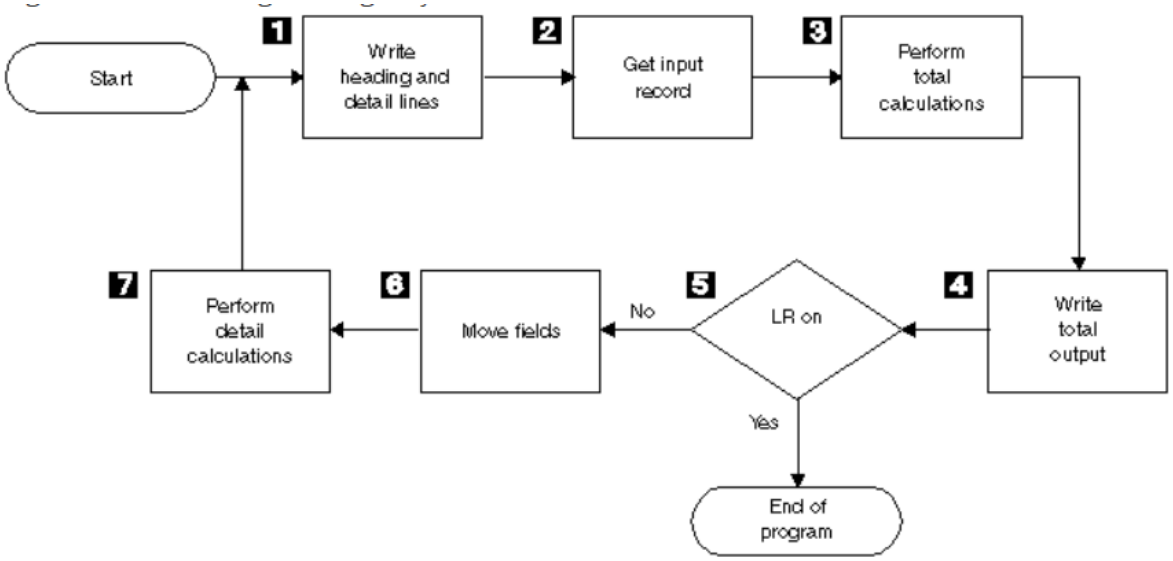
Egy képernyős alkalmazás létrehozásához szükség lehet a display file mellett fizikai file-ra is. Ez a file tartalmazza a tényleges adatot, amivel az RPG program dolgozik, illetve megadja az adatok leírását formailag, gyakorlatilag egy adatbázis fájlként működik. Az SQL szempontjából a fizikai fájl megegyezik a táblával. Emellett a fizikai fájloknak egy vagy több memberük is lehet. A memberek megegyeznek a formátumban, viszont különböző adatokat tartalmaznak. A fizikai fájlban opcionálisan lehet kulcsmezőket is definiálni, amely szerint a rekordok sorba rendeződnek az adatok lehívásakor

A fizikai fájlon felül lehetőség van logikai fájl használatára is, amely nem tartalmaznak adatot, viszont a fizikai fájlhoz kapcsolódnak és meghatározzák a rekord formátumokat, illetve a fizikai fájl kulcsolását is megadhatják.

3.3 RPG logikai ciklus

A RPG programozásnak kulcsfontosságú része az úgynevezett ’RPG Logic Cycle’, vagy magyarul az RPG logikai ciklus, amelynek ismerete a modern RPG fejlesztők számára is elengedhetetlen.

A logikai ciklus tulajdonképpen egy egyszerű működési mód, ami az összes fájlolvasást kezeli és minden RPG program ilyen logika szerint fut le. A működését az alábbi diagram szemlélteti.



3.2 ábra: Az RPG IV logikai ciklusa, Forrás: [9]

1. Az összes heading és detail sor feldolgozása
2. A következő bemeneti adat beolvasása és a control szintű indikátorok on-ra állítása
3. A számítások elvégzése. Ezek L1- L9 illetve LR (LastRecord) alapján vezéreltek
4. Az összes kimeneti sor feldolgozása
5. LR értékéne vizsgálata, ha on, akkor a programnak vége van
6. Az összes bemeneti rekord athelyeződik a processing area-ra. Indikátorok on-ra állítódnak.
7. Az összes detail számítás elvégzése

A 3. lépés tulajdonképpen a programozó által megírt program (Calculation Specification), amelynek a keretét határozza meg az RPG logikai ciklus.Ezen kívül létezik egy részletesebb leírása a logikai ciklusnak 47 lépéssel, ezt szakdolgozatomban nem részletezem.

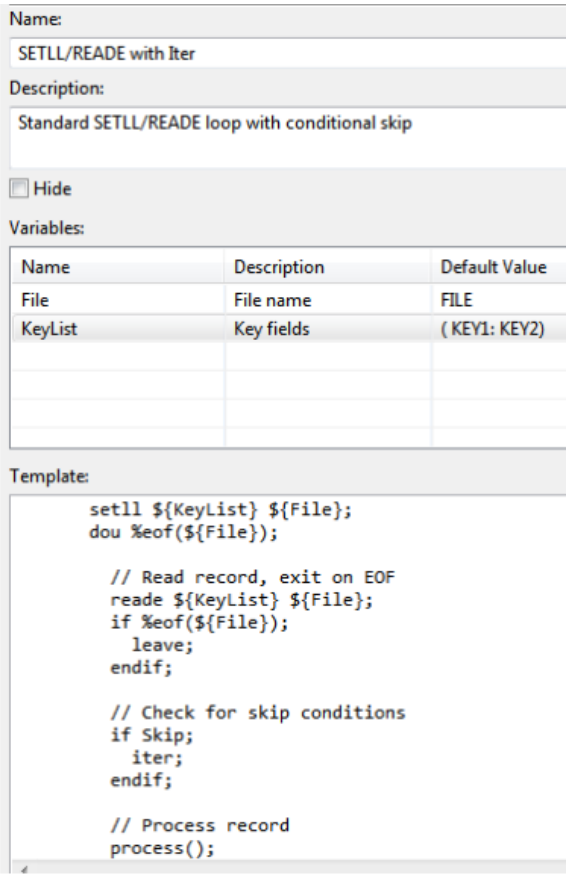
3.4 Snippetek

Szakdolgozatom során fő feladatom egy snippet könyvtár létrehozása volt, melynek célja a szoftverfejlesztés produktivitásának növelése, illetve a fejlesztők munkájának segítése. A snippetek létrehozásának lehetőségét az új fejlesztő környezet, a Rational Developer for i teszi lehetővé, amely egy jelentős fejlődés a korábban használt zöld képernyős környezethez képest és számos új eszközt tartalmaz a korábbi alternatívához képest.

A snippet tulajdonképpen egy sablonhoz hasonló funkció a Rational Developerben, ahol egy előre megírt kódrészletet snippetként elmenthetünk és később újra felhasználhatunk egy másik programban, úgy, hogy ugyanazt a funkciót lássa el (pl. inicializálás), de a funkció megtartása mellett illeszkedjen az adott programhoz, illetve az ott definiált változókhoz. Ezt konkrétan úgy lehet megvalósítani a snippet Managementben, hogy az adott kódrészletben változóként menthetjük el az amúgy program specifikus részeket, pl. fájl- és rekordnevek, bizonyos kulcsszavak, konstansok stb. Majd az új programba beillesztjük az adott snippetet és megadjuk a változók értékeit, pl. a display file vagy a fizikai file nevét, és a Snippet Management automatikusan elvégzi az egész kódrészletben a változásokat.

3.5 Snippetek előnyei

A snippetek használatának számos előnye van a szoftverfejlesztés során. Egyik legfontosabb a produktivitás növelése, ugyanis a snippet segítségével a program írása sokkal gyorsabban történik, mivel az RPG programozás során számos esetben kell ugyanazokat a funkciókat ellátó részeket új egy programba beültetni, gondolok itt a bemeneti file-ok specifikációjára, a subfile-ok inicializálásra, rekord módosító szubrutinokra stb. Ezeket a részeket a Snippet Management segítségével gyorsan beilleszthetjük (lásd 3.3 ábra), és a specifikus részek változtatását automatikusan elvégzi, úgy, hogy illeszkedjen az ott definiált változókhoz. Az így elkészült programok tulajdonképpen moduláris felépítésük és az efféle struktúra nagyban javítja a későbbi karbantartást.



3.3 ábra: snippet beillesztése

Mindazonáltal a kódírás sebességének növelésén túl konzisztenciát is biztosít a fejlesztő csapat számára, ugyanis, ha mindenki ugyanabból a könyvtárból dolgozik, akkor a későbbi programmódosítások és javítások is egyszerűbbek és átláthatóbbak. Gyakran fordul elő, hogy egy projekten többen dolgoznak, így amikor átkerül egy másik programozóhoz a kód, akkor hatékonyabban tud dolgozni, mintha egy új, nem általános módszereket használó kódot kapna.

4 Általános karbantartó program létrehozása

A szakdolgozatom során egyik fő feladatom egy Unicredit Servicesnél használt általános karbantartó alkalmazás fejlesztése, amely segítségével később bemutatom az általam létrehozott snippet könyvtár alkalmazását. A karbantartó alkalmazás fejlesztésekor az egyik kikötés az volt, hogy nem jeleníthetek meg az Unicredit Serviceshez kapcsolható bármilyen információt (pl. üzleti logika, magánszemélyek banki adatai), így egy általános alkalmazás fejlesztése mellett döntöttem. Mindazonáltal ez a karbantartó alkalmazás jól bemutatja az IBM i-t használó cégeknél az alkalmazásfejlesztés követelményeit, illetve nehézségeit is, illetve olyan funkciókat, amelyek tipikusan RPG programozási nyelven valósítanak meg.

Az alkalmazás fejlesztésekor szem előtt kellett tartani, hogy a később létrehozott snippet könyvtár alkalmas legyen bármely karbantartó alkalmazás létrehozására, így minden fő funkcióhoz szükség volt egy kellően nagy snippet gyűjteményt létrehozni. Emellett egy speciálisabb funkciót megvalósító alkalmazás nem lett volna alkalmas arra, hogy demonstrálja a snippet könyvtár használatának előnyeit.

4.1 Igény a snippet könyvtárra

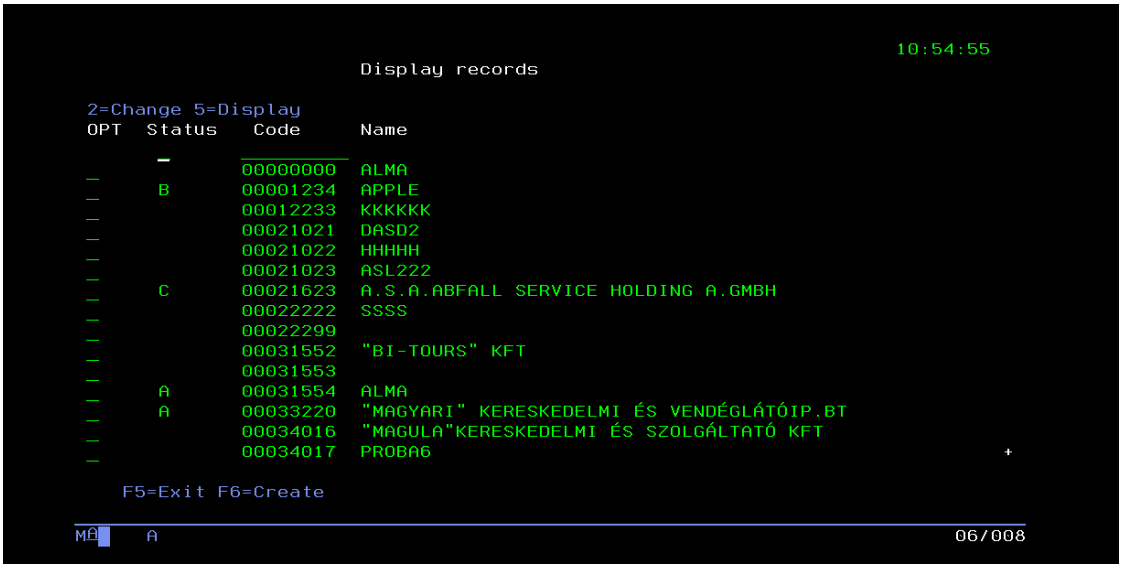
Fontos megjegyezni, hogy az Unicredit Servicesnél, és más hasonló banki informatikával foglalkozó cégnél, rendkívül fontos, hogy az adott projektben meghatározott munka (pl. egy alkalmazás fejlesztése vagy módosítása) időre kész legyen. Egy adott projektre vonatkozó munka időtartamának meghatározása a Business Analyst feladata, azonban a szükséges idő becslése nem mindig pontos és előfordulhat, hogy a tényleges fejlesztés ideje a specifikációban megadott időnél több nappal eltér. Ezért is kulcsfontosságú, hogy minél gyorsabb és gördülékenyebb legyen a fejlesztés, annak érdekében, hogy elkerüljük a projekt csúszását, ami természetesen költséget jelent a cég számára. Természetesen a cégünk is a projekt elvégzéshez szükséges napok számától (Mandays) függően határozza meg a projekt költségét, így egy-két nappal csökkentve az átlagos projektekhez szükséges időt jelentősen növelné a bevételeket. Erre rendkívül alkalmas a snippet könyvtár használata, illetve, ahogy korábban is

említettem a fejlesztőcsapat munkájában is konzisztenciát biztosít, ami szintén hasznos abban az esetben, amikor többen dolgoznak egy projekten.

Továbbá a snippet könyvtár segítségével létrehozott moduláris és strukturált alkalmazás karbantartásához is kevesebb idő szükséges és a debuggolás is könnyebb. Manapság sok cég törekszik az alkalmazások modernizálására az IBM i platformon, szakdolgozatom során én is ezzel a céllal használtam a Rational Developer nyújtotta lehetőségeket.

4.2 Az alkalmazás funkciói

Ahogy az RPG neve is jelzi, a program fő célja eredetileg riportok generálása volt és manapság is gyakran használják ilyen célra. Az általam létrehozott alkalmazás egy adatbázisban lévő adatok megjelenítésére, szerkesztésére, validálására illetve létrehozására szolgál. Maga az alkalmazás viszonylag egyszerű, mégis alkalmas az RPG fő funkcióinak és tulajdonságainak a bemutatására, mint pl. a subfile kezelés, indikátorok használata, illetve a display és fizikai fájl használata, hogy néhányat megemlítsek. Továbbá az alkalmazás ezen részei szinte minden RPG alkalmazásban jelen vannak, így alkalmas a snippet könyvtár létrehozására. Az alkalmazás kezdőképernyője a 4.1-es ábrán látható.



4.1 ábra: Display records kezdőképernyő

Fontos megjegyezni, hogy egy ilyen képernyős alkalmazás létrehozásához egyedül nem elég az RPG kód, szükség van egy vagy több Display fájlra is, ahhoz,

hogy minden egyes felülethez (szerkesztés, megjelenítés, létrehozás) egy képernyőt biztosítsunk, és ennek létrehozás meglehetősen időigényes.

Az alkalmazás kezdőképernyőjén az OPT (Option) mezőnél látható aláhúzott pozícióra kell írni a lehetséges opciókat (2=Change, 5=Display) az adott rekordhoz a felhasználónak. Emellett a Status és Code alatt lévő aláhúzott mezők segítségével adott rekordra lehet szűrni. A megjelenített adatokat nem statikusan másoltam, hanem subfile segítségével egy görgethető felületet hoztam létre, amely az RPG programozás egyik fontos része, az Unicredit Services használt alkalmazásokban is gyakran alkalmazzák. A subfile pontos használatát és alkalmazásának nehézségeit később tárgyalom. Ezen felül több validálási műveletet is el kellett végezni például annak érdekében, hogy a felhasználó biztosan helyes adat hoz létre (pl. nem ad meg üres kód mezőt), ezeket is később részletesen kifejtem.

4.3 A felhasznált adatbázis

Az alkalmazás igényelt egy adatbázist, amit az AS400-as rendszeren egy fizikai fájl tartalmaz. Átalában a fizikai fájlokban bankszámla adatokat tárolunk, természetesen ilyen adatokat nem jeleníthettem meg a szakdolgozatomban, ezért egy általános adatokkal kitöltött adattáblát használtam. A táblának a mezői a customer status, client number, client name, és client address volt.

Ahogy már korábban említettem az IBM i lehetővé teszi az SQL használatát, amely az adatbázis kezeléshez használt nyelv. Egyetemi tanulmányaim során lehetőségem volt megtanulni az SQL használatának alapjait, ami a cégnél végzett más munkáim során sokat segített.

Ahhoz, hogy megnézzük egy adott fizikai fájl tartalmát a PDM-ben az STRSQL parancsot kell használni, majd meg kell adni magát az SQL utasítást. Az esetemben a „SELECT * FROM MYCUSTDTA” parancs a következő táblát adta, amely az alkalmazásom adatait tartalmazta.

4.2 ábra: Adatbázis

4.4 RPG program

A modern RPG programozás egyik követelménye, hogy a kód RPG IV-ben legyen írva a korábban megszokott fixed form helyett free formban [3]. A cég már több, mint 20 éve fejleszt RPG nyelven írt alkalmazásokat, így még mindig rengeteg program fixed form verzióban van és gyakran az ezekből másolt kódrészletek miatt az újabb programok is tartalmaznak ilyen részeket, főleg a F és D specifikációknál. Emiatt a programozás során az F és D specifikációkat a szokásos fixed formban hagytam, viszont a C specifikáció esetén áttértem free formra. Egyetemi tanulmányaim során nem találkoztam RPG programozással, a képzés során főleg a C/C++, MATLAB, illetve Assembly programozást ismertem meg, így teljesen kezdő RPG programozóként kezdtem a munkámat a Unicredit Servicesnél. Ebből kifolyólag teljesen az alapoktól kezdve kaptam oktatást az AS400-as rendszerről, illetve az RPG programozásról is, így a továbbiakban részletesen bemutatom az RPG programozás lépéseit.

4.4.1 File specification

Az alkalmazáshoz írt RPG program az F kártyával kezdődik, ezt a 6. pozícióba írt F betűvel lehet jelölni. Itt kell megadni, hogy melyik adatbázis fájlt használja a program, esetemben ez a MYCUSTDTA (My Customer Data) volt. A neve mellett meg kell adni, hogy az adatbázis frissíthető legyen (u – updatable), illetve, hogy külső adatfájlóról (e - external) van szó.

Az adatbázis fájl mellett meg kell adni a display fájlt is, ez a MYCUSTDTAD volt. A display fájlokat jellemzően úgy nevezzük el, hogy a végén egy D betű legyen. A fizikai fájlhoz hasonlóan a display fájl is külső fájl. Szükség van a display fájlhoz egy

indikátor mező definiálására is, ezt az IndDS kulcsszóval lehet megadni, illetve zárójelben a saját indikátormező nevével.

Az összetettebb alkalmazások gyakran használnak egy vagy akár több subfile-t is, az alkalmazásom esetén egy subfile elég volt. Ezt szintén az F kártyán kell definiálni a SFile kulcsszóval, illetve a subfile rekordnév és subfile rekord number megadásával, ahol az utóbbi a Subfile rekord számának tárolására szolgáló változó. A három definícióhoz szükséges RPG kód a következő.

```
     F.....................................Keywords++++++++++++++++++++++++++
     fMYCUSTDTA uf a e           k Disk
     fMYCUSTDTADcf   e             WorkStn IndDS( DspfInds )
     f                                     SFile( MYCUSTDTAB : SflRrn )
```

4.3 ábra: F kártya

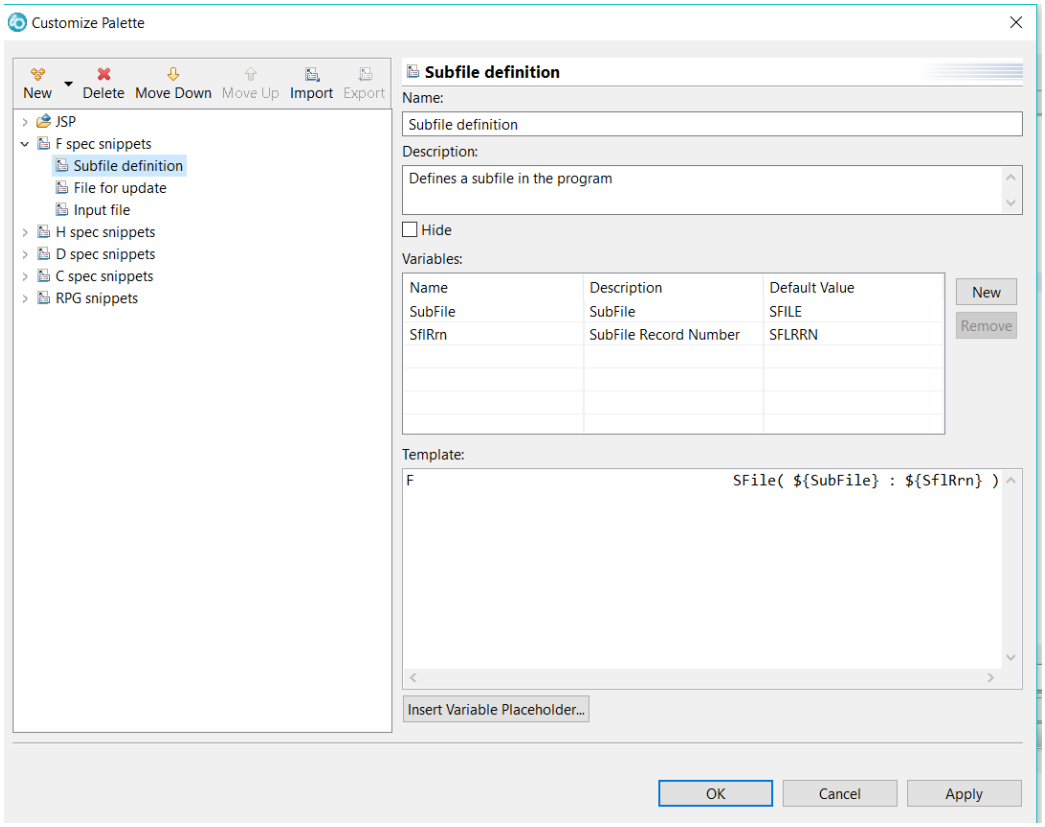
A fenti ábrán látható, hogy egy fájl pontos definiálására több karakter(u,a,e,f), illetve kulcsszó (Inds,SFile) szükséges, amelyek megjegyzése és pontos pozíciójuknak ismerete nehézkes. Továbbá szinte minden RPG programban (pár speciális esetet kivéve) szükség van display fájl, fizikai fájl, illetve subfile definícióra és új program írásakor a programozók gyakran más korábbi programból másolják át ezeket a részeket, átírva az adott fájlra specifikus részt (pl. fájlnév). Emiatt felmerült annak lehetősége, hogy ezt a lépést a Rational Developer nyújtotta Snippet Management segítségével meggyorsítjuk azáltal, hogy létrehozunk egy előre definiált snippet gyűjteményt, amely segítségével létrehozott program moduláris felépítésű lesz. Sok cég tűzte ki célul az RPG nyelvű alkalmazásfejlesztés modernizálását, és ennek egyik módja a programozás során a moduláris felépítés. A következő fejezetben egy konkrét példán keresztül bemutatom egy snippet létrehozását.

4.4.2 Snippet létrehozása

Snippet létrehozására az eredeti zöldképernyős felületen nincs lehetőség, egyedül a Rational Developer képes erre. A fejlesztőcsapatunkban is még folyamatosan történik az átállás az új fejlesztő környezetre, így ennek megismerése, illetve tanulmányozása is része volt a munkámnak. A Snippet Management használata viszonylag egyszerű, ennek ellenére alkalmazása számos előnnyel jár.

Első lépésként az RD Window/Show view menüjéből ki kell választani a snippets-et, ugyanis ez nem alapértelmezett. Az itt létrehozott snippeteket csoportosíthatjuk, természetesen egy nagyobb gyűjtemény esetén merül fel, hogy milyen logika alapján. Lehetőség van pl. F, D, C stb. kártyák szerinti csoportosításra, munkám során én ezt a megoldást választottam.

Ezután egyszerűen kijelöljük a kódszerkesztőben azt a részletet, amit snippetként szeretnénk használni, majd átmásoljuk a snippet view-ba, a Paste as snippet-et kiválasztva. Ezt egy konkrét példán keresztül mutatom be.



4.4 ábra: Subfile definition snippet létrehozása

Ahhoz, hogy a korábban bemutatott Subfile-t később snippetként használjuk meg kell adni a változókat. Ez a variables fülnél lehetséges, a változó nevének és leírásának megadásával, illetve lehetőség van egy alapértelmezett érték megadására is, a felület a 4.4 ábrán látható. A Subfile definiálásakor a ${válozónév} szintaxis segítségével tudjuk jelezni, hogy az adott rész program specifikus, a többi része a kódnak változatlan formában fog megjelenni a snippet használatakor. A Subfile snippet esetén ez csak a SFile kulcsszó volt, a Subfile neve és a Subfile record number

természetesen minden program esetén más, illetve a description jelzi a programozónak, hogy mit jelölnek ezek a változók.

Ezután az utolsó lépés a snippetnek olyan nevet adni, amellyel könnyen azonosítható az általa ellátott funkció, illetve egy rövid leírás, ami további információt ad a programozónak. Annak érdekében, hogy a használata minél gyorsabb legyen fontos, hogy logikus felépítésű snippet könyvtárat hozzunk létre, amelyben a programozó könnyen tud navigálni. Számomra a specifikációkénti csoportosítás tűnt a leglogikusabb választásnak egyelőre, így ez a snippet az „F spec snippets” nevű mappába került.

4.4.3 File Specification snippetek

Az F kártya snippetei rendkívül hasznosak, ugyanis számos esetben kell specifikus kulcsszót használni, amit gyakran az reference guide-ból kell kikeresni, és ez meglehetősen időigényes.

Röviden felsorolva a létrehozott snippeteket:

- Input File definition
- Output File definition
- Subfile definition
- Printer File definition (egyszerű lekérdezéshez)

4.5 Definition Specification

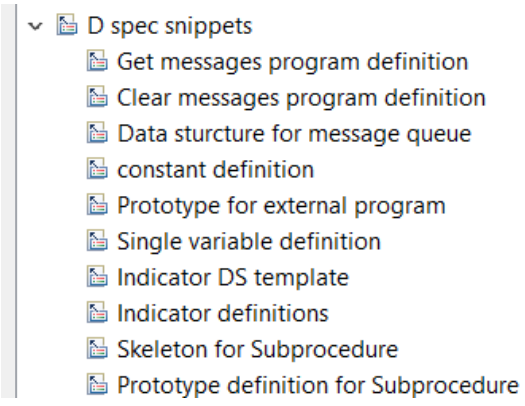
Az File Specification után következő rész az RPG programban a Definition Specification. Itt kerül sor a programban használ változók, adatstruktúrák definiálására, illetve a display fájlhoz tartozó indikátorok definiálására is. A programhoz szükséges változók és indikátorok a 4.5-ös ábrán láthatók.

```
D DspfInds        DS
D  keyf1                         N   Overlay( DspfInds: 01 )
D  keyf3                         N   Overlay( DspfInds: 03 )
D  keyf4                         N   Overlay( DspfInds: 04 )
D  keyf5                         N   Overlay( DspfInds: 05 )
D  keyf6                         N   Overlay( DspfInds: 06 )
D  keyf7                         N   Overlay( DspfInds: 07 )
D  nextone                       N   Overlay( DspfInds: 19 )
D  protected                     N   Overlay( DspfInds: 20 )
D  notvalid                      N   Overlay( DspfInds: 22 )
D  notvalid2                     N   Overlay( DspfInds: 27 )
D  confirm                       N   Overlay( DspfInds: 23 )
D  confirm2                      N   Overlay( DspfInds: 24 )
D  pageup                        N   Overlay( DspfInds: 25 )
D  pagedown                      N   Overlay( DspfInds: 26 )
D  SflDspCtl                     N   Overlay( DspfInds: 91 )
D  SflDsp                        N   Overlay( DspfInds: 92 )
D  SflClr                        N   Overlay( DspfInds: 93 )
D  SflEnd                        N   Overlay( DspfInds: 94 )
D  correct                       N   Overlay( DspfInds: 95 )
D  firstread                     N   Overlay( DspfInds: 96 )
D  from                          N   Overlay( DspfInds: 97 )
D  SflRrn         S             5P 0
D  recnum         S             2P 0
D  status         S             1A
D  @Eof           S              N
D  @DspAtrDP      C                  Const(X'20')
D  @DspAtrRI      C                  Const(X'21')
D  @DspAtrUL      C                  Const(X'24')
D  sMyCUSTDTAF    E Ds               ExtName(MYCUSTDTAD:MYCUSTDTAF)
D                                    Qualified
```

4.5 ábra: a program indikátorai és változói

Az RD-nek köszönhetően a nevek hosszára nincs kikötés, így sokkal beszédesebbek, mint a régi környezetben. A keyf1-keyf7 indikátorok az F funkció billentyűk értékeinek megfelelő on vagy off értékű változók. Itt az N betű jelzi, hogy indikátorról van szó, de emellett lehet még S (single variable) betűvel jelölni a sima változókat, illetve C betűvel a konstansokat. A változóknak típust is lehet adni, a SflRrn esetén például az „5P 0” jelöli, hogy 5 hosszú pakolt decimálisról van szó, 0 tizedes értékkel, vagyis az értéke 0 és 99999 között mozoghat. Indikátorokból összesen maximum 99 lehet, ugyanis az File Specificationben definiált indikátor mező mérete 99, illetve minden indikátorhoz tartozik egy számérték is, amelynek szerepe a display fájl esetén merül fel.

A létrehozott snippeteket a 4.6-os ábra szemlélteti.



4.6 ábra: D spec snippets

4.5.1 Display fájl

Ahhoz, hogy az alkalmazásunk egy adott képernyőn fusson, egy display fájlra van szükség, amit DDS segítségével adhatunk meg. A DDS hasonlóan az RPG-hez rengeteg kulcsszót használ, ezt az IBM egy 300 oldalas guide-ban foglalta össze, én a következőben ismertetem az alkalmazásomhoz szükséges főbb elemeket. A fájlban lehetőség van úgynevezett rekordok megadására, amelyek a képernyő egy bizonyos részét reprezentálják (pl. header, subfile, funkciósor). Annak érdekében, hogy a képernyőn az adott fejlécet láthassuk, a következő kódra van szükség.

```
A          R MYCUSTDTAA
A                                      OVERLAY
A                                    1 70TIME
A                                    2 25'Display records'
A                                      DSPATR(HI)
```

4.7 ábra: Fejléc megadása DDS-sel

Az RPG-hez hasonlóan itt is figyelni kell az egyes kulcsszavak pozíciójára, ugyanis azok csak meghatározott helyen érvényesek. Emellett az RD itt is külön színnel jelzi a kulcsszavakat, illetve rekord neveket, ami a SEU-ban eddig nem volt elérhető. A fejléc rekordneve a MYCUSTDTA, ami egyszerűen abból áll, hogy kiírja az időt (TIME kulcsszó) az első sor 70. oszlopába, illetve kiírja a 'Display records' sztringet a második sor 25. oszlopába.

A kiírt szövegnek emellett tulajdonságokat is adhatunk a DSPATR kulcsszó segítségével, esetemben ez DSPAT(HI) volt, amivel fehér színű lett (az alapértelmezett zöld helyett). Egy display fájlban természetesen több rekordot is megadhatunk, itt fontos megjegyezni a subfile rekordot, ami egy kicsit speciálisabb.

Amellett, hogy a képernyőn megjelenik egy szöveg, amit indikátorok segítségével ki és be lehet kapcsolni, szükség van egy olyan részre, amin az adatokat léptetni vagy görgetni lehet. Ezt a gyakorlatban subfile segítségével oldják meg.

4.5.2 Subfile kezelés

A subfile-okat az adatbázisokhoz hasonlóan lehet definiálni. Lehetőség van olvasni, írni, illetve frissíteni az adatokat az ugyanúgy, mint az adatbázisokban, viszont itt az adatok csak ideiglenesen tárolódnak.

Elsősorban egy subfile rekordra van szükség, ami tartalmazza a megjelenítendő mezőket, ugyanolyan néven, ahogy azok a fizikai fájlban szerepelnek (lásd 4.8 ábra).

```
A          R MYCUSTDTAB
A                                      SFL
A            CUSOPT         1A  B  7  2
A  22                                  DSPATR(RI)
A            CUSSTS         1A  O  7  8
A            CUSCODE        8  0O  7 15
A            CUSNAME       50A  O  7 25
```

4.8 ábra: Subfile rekord

A subfile-ban az option, status, code és name mezők jelennek meg, a kódban látható pozíciókban illetve mezőtípusban (1A – egy karakter, 8 0 – nyolc decimális). Ezenfelül érdemes megjegyezni, hogy mivel ez a kezdőképernyő csak az adatok megjelenítésére szolgál, ezért csak kimeneti (O -Output) adatok, szerkeszteni ezen a képernyőn nem lehet a status, code és name mezőket, csak az option mezőbe írhat a felhasználó (B – both). Az SFL kulcsszó segítségével jelezzük, hogy egy kitüntetett, subfile típusú rekordról van szó.

Az indikátorok használatát jól szemlélteti a CUSOPT-hoz tartozó DSPATR(RI) és a vele egy sorban lévő 22-es jelzés. Abban az esetben, ha a 22-es indikátor értéke on-ra állítódik a programban, akkor érvényesül a vele egy sorban lévő kulcsszó, vagyis az option mező háttere világos lesz, amit a programban arra használunk, ha a felhasználó nem megengedett értéket adott meg.

A subfile rekord mellett szükség van a subfile control rekordra is, amely a subfile működését határozza meg (pl. a megjelenített rekordok számát). Fontos megjegyezni, hogy a DDS írása közben először a subfile rekordot kell definiálni, és csak utána a control rekordot, annak ellenére, hogy a control rekord látható a képernyő felső részén a subfile megjelenítésekor.

```
A          R MYCUSTDTAC                SFLCTL(MYCUSTDTAB)
A                                      CF05(05)
A                                      CF06(06)
A                                      OVERLAY
A                                      SFLSIZ(018)
A                                      SFLPAG(015)
A  91                                  SFLDSPCTL
A  92                                  SFLDSP
A  93                                  SFLCLR
A  94                                  SFLEND
A                                      CSRLOC(HCRROW HCRCOL)
A            HCRROW         3S 0H
A            HCRCOL         3S 0H
A            STSSEARCH      1A  B  6  8
A                                      DSPATR(UL)
A            CDESEARCH      8  0B  6 15
A                                      DSPATR(UL)
A                                   4  2'2=Change 5=Display'
A                                      COLOR(BLU)
A                                   5  2'OPT'
A                                      DSPATR(HI)
A                                   5  7'Status   -
A                                      Code     -
A                                      Name'
A                                      DSPATR(HI)
```

4.9 ábra: Subfile control rekord

Az subfilehoz szükséges subfile control rekordot a 4.9-es ábra szemlélteti. Négy rekord szintű kulcsszót kötelező megadni minden subfile control rekordban, ezek a SFLCTL, SFLDSP, SFLPAG, SFLSIZ.

- Az SFLCTL kulcsszó határozza meg a korábban definiált subfile működését. Itt meg kell adni a subfile rekord nevét.
- Az SFLDSP jelzi a rendszernek, hogy output művelet esetén egy subfile-t kell megjeleníteni.
- Az SFLPAG határozza meg, hogy egyszerre hány subfile rekord jelenjen meg a képernyőn.
- Az SFLSIZ határozza meg, hogy összesen hány rekord szerepelhet a subfileban.

Ezen felül még használtam az SFLCLR, SFLDSPCTL és az SFLEND kulcsszavakat, mindegyiket egy indikátorhoz kötve, ezek a subfile inicializálásához kellettek.

Általában két módszert használnak a subfile-ok betöltésére. Az első az oldalankénti töltés, amikor egy oldalnyi rekord betöltése után végrehajtjuk a subfile controlt. A másik módszer során egyszerre betöltjük az összes adatot a subfileba, majd csak ezután hatjuk végre a subfile controlt, én ezt a megoldást választottam. Az első módszer bonyolultabb, viszont gyorsabb működés érhető el vele, ami rendkívül fontos nagy rekordszámú (több ezer) subfile-ok esetén. A második módszert főként akkor

alkalmazzák, ha viszonylag kevés rekord van, esetemben ez csak pár oldalnyi volt, így lehetett ezt alkalmazni.

Másik fontos feladat az SFLSIZ és SFLPAG értékeinek megadása. Ha a két érték egyenlő, akkor a PageUp vagy PageDown megnyomása esetén a kontrol visszakerül az RPG programhoz. Ha a két érték különbözik, akkor a rendszer maga kezeli a műveleteket, addig léptet, amíg van megjelenítendő adat, és csak akkor adja vissza a kontrolt az RPG programnak, ha elérte a subfile elejét vagy végét. Fontos viszont, hogy ne adjunk túl nagy értéket az SFLSIZ-nak, elkerülve a memória pazarlást.

A CSRLOC kulcsszó segítségével tárolhatjuk el a kurzor pozíciójának az értékét két változóban, a függőleges pozíciót a HCRROW, a vízszintes pozíciót a HCRCOL tárolja. Ennek segítségével tudjuk a programban később automatikusan egy adott helyre pozícionálni kurzort, pl. egy hibásan bevitt adat esetén, ezzel segítve a felhasználót. Ezeket az értékeket nem szükséges megjeleníteni a display fájlban, ezt a H (hidden – rejtett) betűvel lehet elrejteni. Mivel az adott alkalmazáshoz tartozó képernyő felbontása 80*24-es (másik opció 27*132), így elég volt 3 hosszúra megválasztani a két változót.

Az alkalmazásban lehetőség van adott státuszra és code-ra rákeresni, ezeket az STSSEARCH és a CDESEARCH mezők segítségével oldottam meg. Itt meg kellett adni, hogy ez a két mező B (Both) típusú, azaz írni és olvasni is lehet. Továbbá a felhasználónak egy aláhúzással jelöljük, hogy hova kell írni a keresett értéket, ezt a DSPATR(UL) (underline - aláhúzott) kulcsszó megadásával lehetséges. A subfile-ok gyakran tartalmaznak ilyen kereső mezőket, ugyanis előfordul, hogy az alkalmazások több ezer rekordos subfilet jelenítenek meg, így sokkal könnyeben tud navigálni a felhasználó.

Ahhoz, hogy a kezdő képernyőn lehetőség legyen az F5 és F6 billentyű használatára (ami esetemben a kilépés és a létrehozás), a display fájlban meg kell adni az aktivált funkció billentyűket a CF kulcsszó segítségével. Ezeket a funkció billentyűket indikátorhoz is kell kötni (5 és 6), amelyek az RPG programban a keyf5 és keyf6 változókhoz tartoznak és ezeken keresztül lehet vizsgálni, hogy a felhasználó megnyomta-e az adott gombot vagy sem.

A képernyőn látható feliratok megadása rendkívül egyszerű, csupán meg kell adni a kívánt szöveget, illetve annak pozícióját. Az alkalmazásban csupán a táblázat

fejlécére és a lehetséges opciók (létrehozás, változtatás) kiírására volt szükség. Az opciókat kékre állítottam, a feliratokat fehérre, ahogy az AS400-as rendszeren szokás.

4.5.3 Change, Display és New Rekord

A kezdő képernyő, subfile és a subfile control rekord mellett szükség volt olyan képernyőkre, amelyek az adott rekord adatainak kiírását, változtatását, illetve egy új rekord létrehozását teszi lehetővé. A következőkben röviden bemutatom ezeket a képernyőket és az ezekhez tartozó új funkciókat.

```
A          R DISPLAYR
A                                      CF04(04)
A                                  3 25'Display selected -
A                                       record'
A            CUSNAME   R        B  6 11
A                                      DSPATR(PR)
A                                  6  2'Name...:'
A            CUSCITY   R        B  7 11
A                                      DSPATR(PR)
A                                  7  2'City...:'
A            CUSSTREET R        B  8 11
A                                      DSPATR(PR)
A                                  8  2'Street.:'
A            CUSCTRY   R        B  9 11
A                                      DSPATR(PR)
A                                  9  2'Country:'
A            CUSTZIP   R        B 10 11
A                                      DSPATR(PR)
A                                 10  2'Zip....:'
A                                 23  5'F5=Exit F4=Close'
A                                      COLOR(BLU)
```

4.10 ábra: Displayr rekord

Az adatok kiírását a „displayr” rekord teszi lehetővé (lásd 4.10-es ábra), itt megjelenik az adott rekordhoz tartozó többi adat, mint pl. a város, utca, ország és irányítószám. Ezeket az adatokat nem lehet szerkeszteni, ezt a DSPATR(PR) (protected-védett) kulcsszóval állítottam be.

Az új rekord létrehozásához szükséges képernyőn ehhez képest pár kisebb változtatásra volt szükség. Bevezetésre került az ún. program-to-system mező, amely lehetővé teszi pl. az új adatok felvitelekor az ellenőrzést, használatát a 4.11-es ábra szemlélteti.

```
A            NEWPROP        1A  P
A            NEWPROT        1A  P
A            NEWPROR        1A  P
A            NEWPROZ        1A  P
A            CUSNAME   R        B  6 11
A                                    DSPATR(&NEWPROP)
A                                  6  2'Name...:'
A            CUSCODE   R        B  7 11
A                                    DSPATR(&NEWPROT)
A                                  7  2'Code...:'
A            CUSSTREET R        B  8 11
A                                    DSPATR(&NEWPROR)
A                                  8  2'Street.:'
A            CUSCTRY   R        B  9 11
A                                    DSPATR(&NEWPROZ)
```

4.11 ábra: Program-to-system mező

A program-to-system mezőket P betűvel lehet jelölni a display fájlban. Tulajdonképpen úgy viselkednek, mint a változók, az RPG programban lehet állítani az értéküket. Ezen felül viszont hivatkozhatunk rájuk az adott rekordnál megadott kulcsszó paramétereként (pl. DSPATR(&NEWPROP)), ennek eredménye, hogy a rekordhoz tartozó paraméter értékét a programból tudjuk vezérelni.

Az alkalmazásban szükség volt annak ellenőrzésére, hogy új rekord felvitelekor a felhasználó ne adhasson meg üres mezőt. Így amikor ez mégis előfordul, az új rekord létrehozása előtt a program ellenőrzi a bevitt adatokat, és hiba esetén azt jelzi a felhasználónak, az adott rekord kivilágításával. A program-to-system alkalmazása lehetővé teszi, hogy ellenőrzéskor több típusú, specifikusabb visszajelzést adjunk a felhasználónak, pl. különböző színek megadásával, és ezt egyszerűen tudjuk vezérelni a programból. Csupán a különböző beállítások értékeire van szükség, ez elérhető az IBM support oldaláról (lásd 4.12 ábra).

```
D  @DspAtrDP      C                   Const(X'20')
D  @DspAtrRI      C                   Const(X'21')
D  @DspAtrUL      C                   Const(X'24')
```

4.12 ábra: Display attribútumok és a hozzá tartozó értékek hexadecimális formátumban

Ezenfelül aktiválni kellett az F1 funkcióbillentyűt is, ami arra szolgál, hogy a már ellenőrzött adat felvitelét véglegesítse a felhasználó. Az „F1=CONFIRM” felirat viszont indikátorhoz van kötve, így csak akkor jelenik meg, ha már megtörtént az ellenőrzés, ezt a correct indikátor on-ra állításával jeleztem a programban.

Adott rekord adatainak változtatására szolgáló képernyőt az előző képernyőhöz hasonlóan hoztam létre.

Egy AS400-on futó interaktív alkalmazás többféle módon is tud hiba üzenetet küldeni a felhasználónak. A gyakorlatban erre a message subfile-t használják, ugyanis

segítségével részletes információt lehet adni a felhasználónak, akár több hibáról is. A korábban említett subfile-hoz hasonlóan a display fájlban kell definiálni az ehhez szükséges rekordokat.

```
A          R MSGSFL                    SFL
A                                      SFLMSGRCD(24)
A            MSKEY                     SFLMSGKEY
A            MSPGMQ                    SFLPGMQ
A
A          R MSGCTL                    SFLCTL(MSGSFL)
A                                      OVERLAY
A                                      SFLSIZ(2)
A                                      SFLPAG(1)
A                                      SFLINZ
A                                      SFLDSPCTL
A                                      SFLDSP
A N22                                  SFLEND
A            MSPGMQ                    SFLPGMQ
```

4.13 ábra: Message subfile

Maga az üzenet, ami hiba esetén megjelenik egy message fájlban van, ezt külön kell létrehozni a CRTMSGF paranccsal. A message subfile használatát az RPG program tárgyalásakor részletezem.

Egy display fájl létrehozása meglehetősen időigényes folyamat, azonban a Rational Developerben elérhető egy a display file design funkció, amely segítségével a képernyő vizuálisan szerkeszthető, nincs szükség a DDS-es leírásra. Ez a funkció nagymértékben lecsökkenti a display fájl létrehozásához szükséges időt (pl. egy nap helyett pár óra elég), ezáltal növelve a produktivitást.

4.6 Calculation Specification

A calculation specification a program fő része, a definition specification utáni rész. Fontos, hogy a program ne csak a specifikációknak megfelelően működjön, hanem jól strukturált és könnyen átlátható legyen, így sokkal egyszerűbb és gyorsabb a program későbbi módosítása és tovább fejlesztése.

A program törzsét az ún. „main line” adja, innen kerülnek meghívásra a különböző funkciókat ellátó szubrutinok (lásd 4.14 ábra). A program végén az inlr indikátor (last record) on-ra állításával, majd a return utasítással jelezzük a program végét.

```
/free
 //*---------------------------------------------------
 //Mainline
 //*---------------------------------------------------
 Exsr Init;
 Exsr Fill;
 Exsr Check;
 Exsr Cycle;

 *inlr=*on;
 Return;
```

4.14 ábra: Main line

Mivel a main line minden szinte minden RPG programban megtalálható, célszerű volt egy ilyen funkciót ellátó snippetet definiálni, ezt a 4.15-ös ábra szemlélteti.

```
//,Initialize the input and constant area of selection screen
  ExSr ${InitScrn};

//,Until Accept/Cancel entered
  DoU F05Exit;

//, Refresh the calculated area of selection screen
  ExSr ${RfsScrn};

//, Display the selection screen
  ExSr ${DspScrn};

//, Process on the selection screen
  ExSr ${PrcScrn};

  EndDo;
```

4.15 ábra: Main line általános felépítése

Az ábra a snippet szerkesztőből lett kimásolva, így jól látható, hogy mely részek jelölnek változót ($ kezdetűek), illetve jól mutatja, hogy milyen funkciókat ellátó szubrutin-ok szükségesek egy karbantartó alkalmazáshoz. A snippet használatakor a programozónak csak meg kell adni a program specifikus szubrutinok nevét, vagy használható egy alapértelmezett érték is.

Az inicializáló szubrutin tipikusan olyan rész, amit érdemes snippetként használni, ugyanis a subfile inicializálása minden programban ugyanolyan (lásd 4.16 ábra).

```
//-------------------------------------------------------------------
//Inicializalo szubroutine
//-------------------------------------------------------------------
Begsr Init;
//clear msg
GRMVMS();
SflDspCtl=*off;
SflDsp=*off;
SflClr=*on;
SflEnd=*off;
SflRrn=0;
Write MYCUSTDTAC;
SflClr=*off;
SflDspCtl=*on;
EndSr;
```

4.16 ábra: Inicializáló szubrutin

Az init szubrutinban még érdemes a message subfilet törölni, ez a GRMVMS függvény hívásával történik.

Ezt követően a Fill szubrutin (melynek része a ReadNext) feladata a subfile feltöltése az adatbázisból. Itt a kereső mezők miatt szükség volt egy segédváltozó bevezetésére (firstread – első olvasás, alapértelmezett értéke on), ugyanis, ha a kereső mezőben volt egy érték, akkor az ott lévő értéknek megfelelő adatokkal kellett feltölteni a subfilet, amúgy pedig minden értéket be kellett olvasni. Ehhez egy általános snippetet hoztam létre (lásd 4.17 ábra).

```
Begsr ReadNext;
SflRrn=0;
Setll *loval MYCUSTDTA;
Read(N) CUSTFMT;
Dow not %Eof(MYCUSTDTA);
If firstread=*on;
  SflRrn+=1;
  Write MYCUSTDTAB;
Else;
  If CUSSTS=STSSEARCH;
    SflRrn+=1;
    Write MYCUSTDTAB;
  Endif;
Endif;
Read(N) CUSTFMT;
Enddo;
SflEnd=*on;
Endsr;
```

```
Begsr ReadNext;
      ${SflRrn}=0;
      Setll *loval ${PFile};
      Read(N) ${PFileRec};
      Dow not %Eof(${PFile});
        SflRrN+=1;
        Write ${SubFile};
        Read(N) ${PFileRec};
      EndDo;
      SflEnd=*on;
      Endsr;
```

4.17 ábra: ReadNext szubrutin és a hozzá tartozó snippet

A subfile feltöltése után fontos annak ellenőrzése, hogy sikeres volt-e a feltöltés, erre szolgál a Check szubrutin (lásd 4.18 ábra).

```
//
Begsr Check;
If SflRrn <> 0;
SflDsp=*on;
Else;
Write Empty;
Endif;
Endsr;
''
```

4.18 ábra: Check szubrutin

A szubrutin csupán ellenőrzi, hogy nullától különböző rekord van-e a subfile-ban, ha nem, akkor a hiba jelzésére egy Empty nevű képernyő jelenik meg. Természetesen ezt a feladatot ellátó snippetet is létrehoztam. A program következő része a főciklus (lásd 4.19 ábra).

```
''
Begsr Cycle;
Dow keyf5=*off;

  Exsr Display;
  Exsr Process;
  If notvalid=*off;
    Exsr Refresh;
  Endif;
  If CDESEARCH <> 0;
     Exsr Init;
     Exsr ReadNextSearch;
     Exsr Check;
  Endif;
  If STSSEARCH='A' OR STSSEARCH='B' OR STSSEARCH='C';

      Exsr Init;
      Exsr ReadNext;
      Exsr Check;
  Endif;
Enddo;
Endsr;
```

4.19 ábra: Főciklus

A főciklus határozza meg a program futását mindaddig, amíg a felhasználó F5-tel kilép a programból. A Display szubrutin-ban kerül sor a képernyő egyes mezőinek a kiírására (fejléc, subfile, lábléc és message subfile). Fontos megjegyezni, hogy a subfile esetén az exfmt utasítást használtam, amely a subfile kiírása mellett átadja a vezérlést a felhasználónak, aki pl. az F6 megnyomásával tud ezután létrehozni új rekordot, és ezután folytatódik a program végrehajtása. A főciklusban kerül sor a code és a status kereső mezőinek az ellenőrzésére is, itt felhasználtam a korábban definiált szubrutinokat, amelyek segítségével, csak az adott státuszú (pl. A) rekordok jelennek meg a subfile betöltésekor.

A Process szubrutinban végeztem el a kezdőképernyő megjelenítése utáni műveleteket, mint pl. az OPT mező ellenőrzése, a kód a 4.20-as ábrán látható.

```
Begsr Process;
GRMVMS();
Select;
  When keyf5;
    *inlr=*on;
    return;
  When keyf6=*on;
    Exsr Newone;
  Other;
  Readc MYCUSTDTAB;
  Dow keyf4=*off AND keyf5=*off AND %eof=*off;
  Select;
    When CUSOPT <>'2' AND CUSOPT <>'5'AND CUSOPT <> *blank;
      notvalid=*on;
      GGETMS('MIC0001');
      Update MYCUSTDTAB;
    When CUSOPT='2';
      notvalid=*off;
      Exsr Chng;
      Clear CUSOPT;
      Update MYCUSTDTAB;
    When CUSOPT='5';
      notvalid=*off;
      Exsr Dsp;
      Clear CUSOPT;
      Update MYCUSTDTAB;
      Endsl;
      Readc MYCUSTDTAB;
   Enddo;
Endsl;
Endsr;
```

4.20 ábra: Process szubrutin

Látható, hogy az F6-os funkcióbillentyű segítségével lehet létrehozni új rekordot, illetve az F5-tel lehetséges a kilépés. Azonban, ha egyik se teljesült, akkor a szubrutin ellenőrzi az OPT mezők értékét, miután a felhasználó bevitte az értéket és entert ütött (az enter lenyomása megfelel az Other ágnak).

A szubrutin létrehozásakor fontos volt, hogy a program több OPT megadása esetén (pl. első három rekordnál 2) ne térjen vissza a kezdőképernyőre, hanem egymás után lehessen a három rekordot szerkeszteni. Emiatt volt szükség még egy ciklusra, illetve a ciklus végén a Readc utasításra, amely beolvassa az OPT mező értékeit.

Egy hibás OPT megadása esetén (ha nem 2,5 vagy üres) szükség volt a hiba jelzésére, amely az OPT mező kivilágítása volt és egy message a felhasználónak a hiba típusáról. A mező kivilágítását a notvalid indikátor on-ra állításával lehetett elérni, ami a korábban display fájlban megadott RI (reverse image) attribútumhoz van kötve.

Továbbá a hibaüzenethez szükség volt az AS400-on megadni egy üzenetet MIC0001 néven, amit a GGETMS program meghívásával lehet megjeleníteni.

Mivel a Process szubrutinra vagy hozzá hasonló programrészletre gyakran van szükség, így létrehoztam egy általános Process snippetet (lásd 4.21 ábra).

```
Begsr Process;
        //message program
        Select;
          When keyF5;
            *inlr = *on;
            return;

          When keyF6;
            Exsr ${NewRecord};

          Other;
            Readc ${SflCtl};
            Dow keyF4=*off AND keyF5=*off AND %Eof=*off;
              Select;
                When ${Option}='2';
                   ${Notvaid}=*off;
                   Exsr ${Change};
                   Clear ${Option};
                   Update ${SflCtl};

                 When ${Option}='5';
                   ${Notvaid}=*off;
                   Exsr ${Display};
                   Clear ${Option};
                   Update ${SflCtl};

              EndSl;
              Readc ${SflCtl};
            EndDo;
        EndSl;
            EndSr;
```

4.21 Process snippet

A Process szubrutinban látható három új szubrutin, a Chng, Newone és a Dsp. Mivel a Chng tartalmaz minden olyan funkciót, mint a többi, így a továbbiakban csak ezt mutatom be.

A Chng szubrutin végzi el a kiválasztott rekord adatainak a szerkesztését (lásd 4.22-es ábra).

```
Begsr Chng;
Chain CUSCODE MYCUSTDTA;
keyf4=*off;
confirm=*off;
correct=*off;
CUSPROP=@DspAtrUL;
CUSPROT=@DspAtrUL;
CUSPROR=@DspAtrUL;
If %found(MYCUSTDTA);
  dow keyf5=*off AND keyf4=*off AND confirm=*off;
  Exfmt CHANGE;
  Select;
  When keyF1;
    Exsr Validate;
    If correct;
      confirm=*on;
      Update CUSTFMT;
    Endif;
  Other;
    Exsr Validate;
  Endsl;
  Enddo;
Endif;
Endsr;
```

4.22 ábra: Chng szubroutin

Először szükség van a kiválasztott rekord megkeresésére a MYCUSTDTA táblából, ez a Chain utasítással lehetséges, ami a code alapján rááll arra a rekordra és beolvassa. Mivel szükség van a bevitt értékek ellenőrzésére a szerkesztés során, szükséges a mezőkhöz tartozó attribútumokat alaphelyzetbe állítani, ezek alapesetben aláhúzottak, ezt a DspAtrUL (Display Attribute Underline) konstanssal állítottam be. Ahhoz, hogy a mezők a képernyőn adott attribútum szerint jelenjenek meg, a display fájlban is meg kell adni az itt definiált változókat, ahogy korábban említettem.

Abban az esetben, ha az adott code szerinti rekord létezik az adattáblában, a program egy ciklusban folytatódik, amelyből F4, F5 billentyűvel lehet kilépni, vagy akkor, ha a szerkesztés után bevitt adatok ellenőrizve voltak, illetve a felhasználó megnyomta az F1 (confirm) billentyűt. Fontos megjegyezni, hogy a felhasználó számára csak akkor aktív az F1 billentyű, ha az ellenőrzés megtörtént, ezt egy indikátorral oldottam meg. Az adattábla frissítése az Update utasítással történik, megadva a tábla rekord nevét (CUSTFMT).

A korábban említett szubrutin fontos része az ellenőrző szubrutin (validate), amelyet gyakran használnak az Unicredit Services-es karbantartó programok esetén. Természetesen, amikor a felhasználó felvisz egy új rekordot szükség van arra, hogy az új rekord bizonyos követelményeknek megfeleljen, pl. adott karakterszámú legyen, ha

bankszámlaszámot ad meg vagy ne adhasson meg üres mezőt ott, ahol kötelezően ki kell tölteni. Erre szolgál a validate szubrutin (lásd 4.23 ábra).

```
Begsr Validate;
Select;
When CUSNAME=*blank;
  CUSPROP=@DspAtrRI;
  correct=*off;
  HCRROW=6;
  HCRCOL=11;
When CUSCITY=*blank;
  correct=*off;
  CUSPROT=@DspAtrRI;
  HCRROW=7;
  HCRCOL=11;
When CUSSTREET=*blank;
  CUSPROR=@DspAtrRI;
  correct=*off;
  HCRROW=8;
  HCRCOL=11;
Other;
  correct=*on;
  CUSPROP=@DspAtrUL;
  CUSPROT=@DspAtrUL;
  CUSPROR=@DspAtrUL;
Endsl;
Endsr;
```

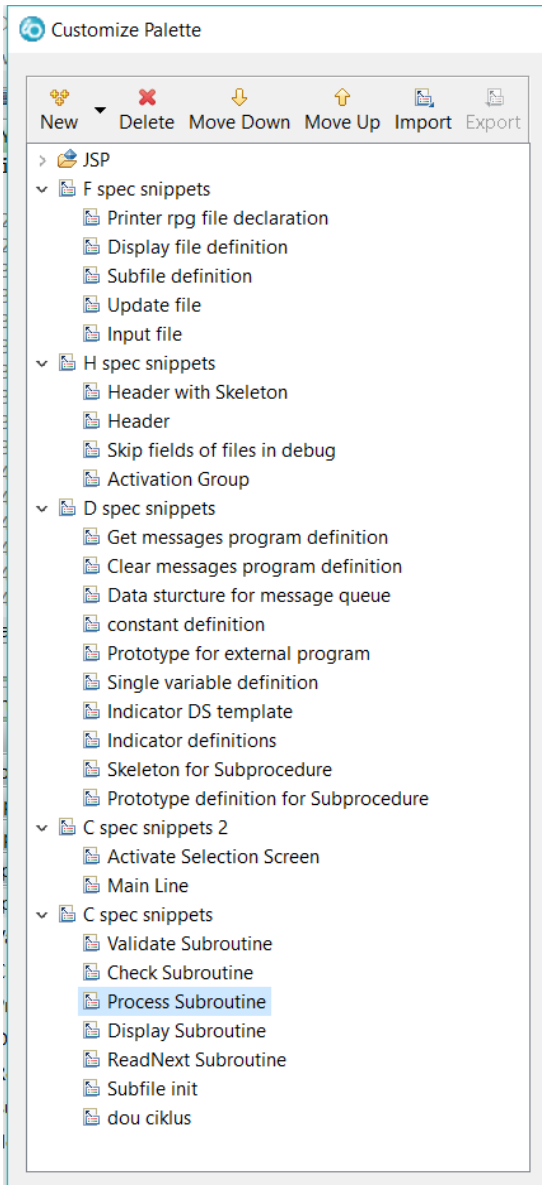
4.23 ábra: Validate szubrutin

Esetemben egyszerűen csak az ellenőriztem, hogy üres-e a kitöltendő mező. Ha üres volt, akkor ezt a mező kivilágításával (reverse image) jeleztem a felhasználónak, illetve a kurzort a mező elejére pozícionáltam felhasználva a HCRROW és HCRCOL változókat. Az ehhez tartozó snippetet a 4.24-es ábra szemlélteti.

```
Begsr Validate;
      Select;
      When ${Field1}=*blank;
        CUSPROP=@DspAtrRI;
        correct=*off;
        HCRROW=${posx1};
        HCRCOL=${posy1};
      When ${Field2}=*blank;
        correct=*off;
        CUSPROT=@DspAtrRI;
        HCRROW=${posx2};
        HCRCOL=${posy2};
      Other;
        correct=*on;
        CUSPROP=@DspAtrUL;
        CUSPROT=@DspAtrUL;
      Endsl;
      Endsr;
```

4.24 ábra: Validate snippet

Végezetül pedig bemutatom az elkészült snippet könyvtárat (lásd 4.25 ábra).



4.25 ábra: Az elkészült snippet könyvtár

Természetesen ez még nem a végleges verzió, de jó alapot biztosít a snippetek későbbi használatához, illetve a továbbiakban a fejlesztés során felmerülő igényeknek megfelelően formálható és bővíthető.

5 Összefoglalás és kitekintés

Annak becslése, hogy a snippetek és a Rational Developer alkalmazásával milyen mértékű produktivitás növelés várható, meglehetősen komplex feladat. A kódírás sebessége több tényezőtől is függ, többek közt a Technical Analyst által elkészített specifikáció részletességétől és minőségétől is, ami a fejlesztő hatáskörén kívül van. A már elkészült kódot a tesztelés után gyakran visszaküldik a fejlesztőknek további módosítások, illetve új funkciók igénye miatt, így a projekt fejlesztési ideje jelentősen megnőhet. Ebből kifolyólag gyakran különbözik a projektre megbecsült fejlesztési idő és a ténylegesen ráfordított idő. A snippetek és a Rational Developer alkalmazása mindazonáltal egy jó kiindulópont annak érdekében, hogy a fejlesztés minél gyorsabb és konzisztensebb legyen.

Az Unicredit Serviceshez pályakezdő RPG programozóként kerültem, így szükség volt a nyelv elsajátítására az alapoktól kezdve. Természetesen az általam elkészített általános karbantartó program egy gyakorlott RPG programozónak sokkal kevesebb időbe telne, így fontos a becslés során az is, hogy mennyire tapasztalt programozóról van szó. Az viszont egyértelmű, hogy a snippetek használata jelentősen lerövidíti egy program megírását egy kezdő programozó esetén is.

Becslésem szerint a létrehozott snippet könyvtár és a Rational Developer segítségével ugyanezt a karbantartó alkalmazást egy kezdő RPG programozó 1-2 nap alatt lekódolhatja, nagyrészt a snippetek strukturált felépítésének, illetve a hozzájuk tartozó leírásnak köszönhetően.

Mindazonáltal az UnicreditServices-nél dolgozó több, mint 10 éves RPG programozási tapasztalattal rendelkező kollégáim véleménye szerint a Rational Developer és a snippetek használata jelenti a jövőt RPG programozás terén. Ahhoz, hogy a snippetekben rejlő lehetőségeket kihasználjuk, szükség van a folyamatos alkalmazására és fejlesztésére, ennek következtében új öltetek és módosítások merülnek fel, ami szintén hozzájárul a produktivitás növeléséhez. Természetesen szükség van arra, hogy a zöldképernyős rendszerhez szokott fejlesztők elsajátítsák a Rational Developer használtát, viszont az egyetemről kikerülő friss diplomások számára ez a modern fejlesztőkörnyezet már ismerős lehet, ezáltal sokkal gyorsabb a betanulási folyamat is.

6 Köszönetnyilvánítás

Végezetül szeretném megköszönni családomnak a folymatos támogatást, ahhoz, hogy ez a szakdolgozat elkészülhessen.

Továbbá szeretném megköszönni egyetemi konzulensemnek, Dr. Renczes Balázsnak a támogatását, útmutatását és segítségét a szakdolgozat felépítésében.

Emellett szeretném megköszönni kollégámnak és külső konzulensemnek, Kern Andrásnak a folyamatos oktatást és segítséget, illetve, hogy felajánlotta számomra ezt a rendkívül érdekes és hasznos témát, amely kapcsán megszerzett tudást további munkám során használhatom az Unicredit Services-nél.

Irodalomjegyzék

[1] Mike Dawson and Marge Hohly: *Understanding AS/400 System Operations*, ISBN: 1-58347-015-8, Mc Press; 1 edition (May 1, 2000)

[2] Enlyft: *Companies using AS/400* https://enlyft.com/tech/products/ibm-as-400

[3] Modernizing IBM i Applications from the Database up to the User Interface and Everything in Between, ISBN-10: 073843986X, IBM Redbooks, 2017

[4] Mcpressonline: *Rational developer for i alternatives* https://www.mcpressonline.com/programming/rpg/rational-developer-for-i-alternatives

[5] Mcpressonline: *Practically Rational: Snippets* https://www.mcpressonline.com/programming/rpg/practically-rational-snippets

[6] ILE RPG Programmer's Guide, IBM, 2019 https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzasc/sc092507.pdf?view=kc

[7] Mcpressonline: RPG academy – modernization: guidelines for modernization goals part 3. https://www.mcpressonline.com/programming/rpg/rpg-academy-modernization-guidelines-for-modernization-goals-part-3

[8] Craig Lutredge's free tools https://www.craigthetechteacher.com/

[9] General RPG IV Logic cycle https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_74/rzasd/gencyc.htm