Cyberflex Java Kártya alkalmazása

Tartalomjegyzék

Bevezető

Az ősember valamikor rádöbbent arra, hogy a két kezén kívül, amely abban az időben talán csak a mellső láb feladatait látta el, eszközök használatával lényegesen könnyebbé teheti életét. Ez a felismerés indította el minden bizonnyal azt a folyamatot, amit a mai tudomány „emberré válás”-nak nevez. Az ember élete könnyebbé vált, és arra is ráért, hogy kísérletezzen, és hogy elkezdjen gondolkodni. Bizonyára hamar ráébredt, hogy egy kis szellemi erőkifejtés árán sok fizikai munkát tud spórolni.

És ettől a pillanattól kezdve csak gondolkodott, és gondolkodott, és …

Nos, hogy ez a folyamat feltétlenül a legjobb irányt vette-e annak idején, annak eldöntésére nem én vagyok hivatott. Az mindenesetre bizonyos, hogy az életük első negyedét megélt fiatal emberi egyedeknek ma nem kellene szakdolgozatot írni, ha ez akkor nem így történik.

Az ember először a legnehezebb fizikai munkától szeretett volna megszabadulni. Eleinte ezt a társadalmi berendezkedés alakítása útján próbálta elérni, hogy tudniillik rabszolgákkal végeztette a nehéz fizikai munkát. Később az újkori ember erőgépeket készített magának, majd kitalálta a számítógépet, amelynek segítségével az erőgépeket automatikusan irányíthatta, és sok más feladatot is könnyedén megoldhatott. Ez néha rosszul sült el, mert olykor a számítógép csak bosszúságot okozott azzal, hogy nem volt képes kitalálni az ember gondolatát. Az ember ezt is megoldotta, amikor kitalálta a programozót, hogy ő nyűglődjön a számítógépekkel. Végül azon is elgondolkodott, hogy a sok gondolkodásba is el lehet fáradni, és megpróbált olyan számítógépet szerkeszteni, amelyik gondolkodni is képes helyette. Ez utóbbi próbálkozás sikertelen volt. Bár ma még nincs olyan gép, aki gondolkodni tudna, mégis ezen a vonalon sok hasznos eredményt értünk el, amelyeken keresztül a számítógépek sok olyan területen is képesek segíteni az embert, ahol nem is olyan egyszerű automatizálni a feladatokat.

Mindazonáltal még mindig megoldatlan az a probléma, hogy az ember ma is sokszor elvesződik a bürokrácia és a töménytelen mennyiségű adminisztrációs feladat között. Ezért a mi „homo cogitus”-unk újra elgondolkodott, és arra jutott, hogy jó lenne, ha mindenkinek lenne egy kártyája, amely minden olyan információt tartalmaz, amit eddig a pénztárcában, az irattartóban, otthon az íróasztalban és egyéb eldugott helyeken (ahonnan gyakran nem kerülnek elő) tartott.

Bár egy kicsit sajátosan fogalmaztam meg a fentieket, de véleményem szerint ezek azok az okok, amelyek megmagyarázzák a kártyás rendszerek terjedését. Ehhez hozzájárul az fizikai eszközök (hardware) fejlődése is. A memória modulok kapacitásának növekedésével, és méretük csökkenésével, valamint a processzorok viharos fejlődésével újabb és újabb lehetőségek nyílnak olyan rendszerek fejlesztésére, amelyek hatékonyan helyettesíthetik a monoton, és ez által automatizálható emberi munkát.

A szakdolgozat célja: egy kicsit elmerülni a kártyás alkalmazások világában, rámutatva a bennük rejlő lehetőségekre. Szeretné(n)k néhány ötletet felvetni az ősszel a felső- és középiskolai oktatásban bevezetésre kerülő diákigazolvánnyal kapcsolatban. Ehhez kapcsolódva szeretnénk egy egyszerű alkalmazáson keresztül bemutatni, hogy hogyan nézne ki ez a gyakorlatban.

A terület alapos és színvonalas bemutatása érdekében ketten készítettük a dolgozatot. A témabejelentőben szereplő témamegosztástól eltérően, a következőképpen osztottuk fel a feladatokat:

- Társam készíti a hálózati alkalmazást. Feladata egy szerverprogram megírása, amely a központilag tárolt adatokat kezeli, valamint a terminálon futó program megírása is, amely a hálózaton keresztül kapcsolatba lép a szerverrel, és kiszolgálja a felhasználókat.
- Az ő alkalmazása az általam készített kártyás alkalmazásra támaszkodik. Nekem kell kidolgoznom és elkészítenem a kártyával kommunikáló programot.

A feladat megosztásával demonstrálhatjuk azt is, hogy mennyire tudunk csapatban (team-ben) dolgozni, hisz a programozói feladatok legnagyobb része nem oldható meg egyetlen személy által.

A kártyás rendszerek általános leírása

A világban egyre elterjedtebbek a kártyás rendszerek, mert a kezelésük egyszerű és kényelmes, mégis szolgáltatások széles tárházát képes nyújtani. Multifunkcionális lehetőségein keresztül mindezeket a feladatokat akár egyetlen kártya is el tudja látni. Mint azt az előző fejezetben vázoltuk, kiválóan alkalmas a kereskedelmi, telekommunikációs, közlekedéssel kapcsolatos és egyéb feladatok ellátására. A fejlesztés a különböző területeken párhuzamosan indult el, ebből kifolyólag a kártyás rendszerek több változata is elterjedt. A következőkben szeretném vázolni ezeknek a rendszereknek néhány közös vonását, majd az intelligens kártyák felépítését tárgyalni.

Fogalmak

Minden kártyás rendszer alapeleme egy szabványos méretű plasztik kártya, amely különböző adathordozókat visel magán (vonalkód, mágnescsík, integrált áramkör). A kártya felhasználásakor szükség van egy olyan eszközre, amely a kártya adathordozójának tartalmát el tudja olvasni, vagy esetleg át tudja írni. Az intelligens kártyák esetén ez az eszköz valósítja meg a kommunikációt a kártyán található processzorral. Azt az eszközt, amely megvalósítja a tulajdonképpeni fizikai kapcsolatot, a továbbiakban „olvasó”-nak fogom nevezni (Card Acceptance Device). Azt a „számítógépet”, amely az olvasó segítségével nyert adatokat feldolgozza, illetve a kártyával történő kommunikációt megoldja, a következőkben „terminál”-nak fogom nevezni. Általában a terminál az, amely megvalósítja a kapcsolatot a felhasználóval, és az ügyfél a terminálon keresztül veheti igénybe a szolgáltatásokat (A felhasználó a terminál billentyűin üti be PIN kódját, jelzi kívánságát, és a terminál szolgálja ki). A fentebb definiált fogalmak sokszor más néven, vagy esetleg más jelentéssel szerepelnek.

Alkalmazások

A kártyás rendszerek a szolgáltató ipar minden területén áttörést értek el, és az egyszerű kezelhetőség és a kényelmes használat miatt egyre nagyobb teret nyernek. A kártyákkal már a magyar vásárlók is megismerkedhettek, és sok sikertelen próbálkozás és kudarc is kísérte ezeket az élményeket. Ezért a magyar közvélemény fenntartásokkal fogadja a kártyák elterjedését. Másként áll ehhez hozzá a nyugati országok polgára. Számára ez e technológia teljesen elfogadott, és az Európai Unió ajánlásaiban is szerepel utalás a korszerű kártyás rendszerek alkalmazására vonatkozóan. A magyar bankok részére a Europay ajánlások mérvadók, melyeket az Uniós csatlakozáshoz kötelezően alkalmazni kell. A fontosabb alkalmazási területek a következők:

- Az elektronikus fizetőeszközök már Magyarországon is széles körben elterjedtek. A különböző bank- illetve hitelkártyák ma már nem hiányozhatnak a bankok által nyújtott szolgáltatások közül. A kártyák nem csak a készpénz kímélése céljából terjedtek el, hanem segítségükkel nagymértékben csökkenthető a kockázat és a kezelési költségek.

- A telekommunikáció terén lassan gyakorlatilag egyeduralkodóvá válik a kártyák használata. Ez különösen igaz a telekommunikáció fiatalabb ágaira, ahol a teljes szolgáltatás egy kártyás rendszerre épül. Ilyen a Magyarországon is elterjedt mobil telefonok piaca. A holland telefontársaság ma egy olyan rendszert vezet be, amely az utcai telefonok segítségével igénybe vehető szolgáltatások körét lényegesen kitágítja. A rendszer koncepciója szerint az utcai telefonfülkéket kiegészítve alkalmassá tehetik multimédiás szolgáltatások nyújtására. Egy egyszerű telefonfülkéből lehetőség nyílik majd az Internet használatára és azon keresztül áruvásárlásra illetve szolgáltatások megrendelésére.
- Egy másik fejlődései irány a fizető tévékészülékek területe. Itt is szeretnék a kártyás rendszer bevezetésével egyszerűbbé és olcsóbbá tenni a különféle szolgáltatásokat, hisz a kártya is alkalmas az előfizetett csatornák vételéhez szükséges adatok tárolására, és huszadannyiba kerül, mint a külön vevők telepítése.
- A közlekedés területén a szomszédos Szlovákia városaiban működik egy rendszer, amely a városi tömegközlekedést könnyíti meg. A szlovák tapasztalatok szerint megnőtt a kihasználtság, és az utasoknak nem kell hosszú percekig a pénztárnál sorban állniuk. Az autópályák használatát is leegyszerűsítheti egy ilyen kártyás rendszer. Erre a célra külön fejlesztették illetve fejlesztik ki azokat a kártyákat, amelyek rádiókapcsolat útján is kommunikálni tudnak az olvasóval, így a kocsiból még csak ki se kell szállni, minden adminisztráció automatikusan és gördülékenyen történik.
- A kártyás fejlesztéseknek egy fontos ága az egészségügy. Az orvosok is felfedezték, hogy milyen kitűnő lehetőségek rejlenek egy egészségügyi kártyában. A rendszer tehermentesítené az orvosi rendelőket a bonyolult adminisztrációs feladatok alól, és így több idő és pénz jutna a betegre. Az egészségügyi kártya ezenkívül tárolhatná a betegre vonatkozó, olykor életfontosságú adatokat, amelyek egy baleset vagy más sürgősségi estben azonnal az orvosok rendelkezésére állna. Az Egészségügyi adatok tárolására kifejlesztettek egy olyan rendszert, amelynek segítségével a kártya adatai könnyedén átfordítható a világ bármely nyelvére, ezzel is felkészülve arra, hogy az adatokat valószínűleg nemzetközi felhasználásra kell tervezni.
- Több országban tervezik a kártyás személyi igazolvány és útlevél bevezetését. Malajzia kormánya tavaly hirdette meg az „egy ember, egy kártya” filozófiát. Amennyiben sikerül a terveket megvalósítaniuk, a bolygónkon ők lesznek az első olyan polgárok, akik a legmodernebb és sokoldalúbb személyi azonosító eszközt hordhatják majd maguknál. Legfrissebb értesülések szerint adatvédelmi aggályok miatt az egészségügyi, banki és személyazonosítási (államigazgatási) feladatokra három különböző kártyát vezetnének be. Távlati célként - a társadalom kártyás rendszerekbe vetett bizalmának növekedése után - a három kártya egyesítése is elképzelhető.

Az eddigiekből is kiderül, hogy az intelligens kártyákban milyen sok lehetőség rejlik. Az intelligens kártya tekinthető a következő évezred azonosítási technológiájának.

Osztályozás

A kártyán található információk tárolási módja szerint létezik:

- Vonalkódos azonosítás esetén a kártyán egy hagyományos vonalkód jelzi a tulajdonos személyét. Ilyen például a könyvtári belépő.
- Mágneskártyák esetén egy mágnescsík tartalmazza az információt. Amikor a kártyát az olvasó előtt elhúzzuk, az leolvassa a kártyát azonosító kódot. A mágnescsík tartalmazhat ezenkívül egyéb adatokat is, amely alapján tulajdonost azonosíthatja. Ilyen kártyák terjedtek el a banki alkalmazások területén, ahol a nyilvános ATM automaták csak a kártya azonosítóját ill. a számlaszámot olvassák le e kártyáról. Ez után az automata kapcsolatba lép a bank központi számítógépével, és az ott tárolt adatok alapján ad készpénzt az ügyfélnek.
- Az integrált áramkörös memóriakártya esetén (IC kártya) a műanyag lapba egy integrált áramkör van beágyazva, amely a memória modulokhoz hasonlóan működik. A tárolt adatokat az olvasó a nyolc csatlakozója segítségével tudja leolvasni. Itt a számolási műveleteket az olvasó végzi, a kártya csak az adatokat tárolja. Ilyen kártyákat használ a Magyarországon bevezetett telefonkártya rendszer.
- Az intelligens kártyák (angolul „smart card”, a magyar „intelligens kártya” elnevezés helyett sok helyen használják a magyarosított „chip kártya” elnevezést, ami jobban utal a kártya felépítésére) esetén a műanyag lapba egy processzor van beépítve, így ez a kártya az adatok tárolásán kívül, különböző vezérlési műveletek elvégzésére is alkalmas. Mivel a kártyának saját memóriája, processzora van, ami a működéshez szükséges feszültséget a lábain keresztül kapja, ezért gyakorlatilag egy önálló, programozható számítógépnek is lehet tekinteni. Így könnyebben megoldódnak a különböző biztonsági problémák, hisz a kártya maga megvizsgálhatja az adatok kiszolgáltatásakor, hogy a kérést intéző jogosult-e az információk olvasására. A kártyán tárolt adatokhoz a kártya beleegyezése nélkül semmilyen módon nem lehet hozzáférni. Így a különböző titkosítási algoritmusok segítségével, az adatokat majdnem tökéletesen lehet védeni. A titkosító algoritmusok támogatása céljából sok kártyába külön kriptoprocesszort is beágyaznak, amelyek hatékonyan és gyorsan képesek nagyobb műveletigényű titkosító algoritmusok végrehajtására is.

Az eddig tárgyalt kártyatípusok közül az első négy esetén a kártya csak adatot tárol, ezért őket passzív kártyáknak nevezzük. Az intelligens kártya maga is képes műveletek végrehatására, ezt jelenti az aktív kártya megnevezés.

A kártyás rendszereket osztályozhatjuk még a következőképpen is:

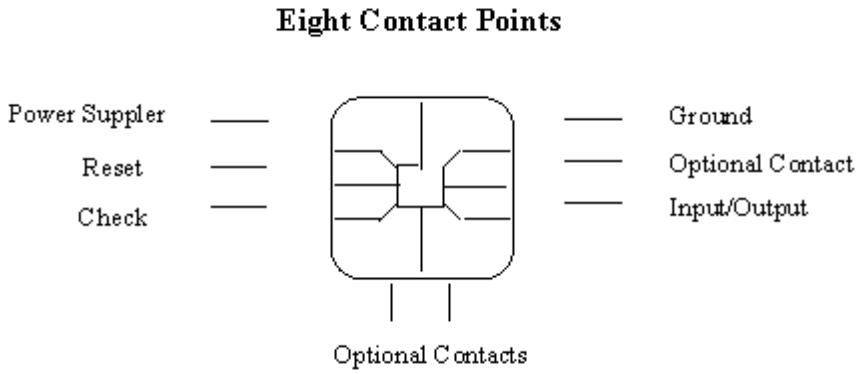
- Léteznek zárt rendszerek, amikor a kártya úgymond „belső használatra” készült, és alkalmazási területének határai pontosan meg vannak határozva. Ilyenek a különböző beléptető rendszerek vagy valamilyen olajtársaság által kiadott kártyák, amelynek segítségével a társaság benzinkútjainál lehet vásárolni. Ezek a rendszerek általában egy szűk, jól meghatározott területen old meg valamilyen adminisztrációs feladatot.

- A nyílt rendszerek sajátossága, hogy alkalmazási területük nincs pontosan behatárolva, és legtöbbször a szolgáltató cégek közötti egyességen alapul. Ilyen kártyák esetében fontosak a szabványos megoldások, mert a szabványok segítségével a kártya felhasználási köre jelentősen tágítható.

Szabványok

A kártya különböző paramétereit és funkcióit a Nemzetközi Szabványügyi Hivatal (ISO) határozza meg. A kártyára vonatkozó szabványok az ISO Standard 7816 rögzíti. Ennek a szabványnak hét része van:

- ISO 7816-1 határozza meg a fizikai paramétereket. Ez tartalmazza, hogy a kártyának mekkorának kell lennie. (szélesség: 85,72 mm/3,375 inch, magasság: 54,03 mm/2,125 inch, vastagság: 0,76 mm (±0,08)/0,03 inch)
- ISO 7816-2 írja le a kártya kapcsolódási pontjait, azaz a kártya nyolc lábának a szerepét.



  - clock, azaz órajel
  - reset, azaz törlőjel
  - 0 V-os láb, ami a földelésnek felel meg
  - 5 V-os láb a tápfeszültségnek
  - 25 V-os láb a programozáshoz szükséges feszültség részére
  - I/O láb, amin az adatjelek közlekednek
  - 2 tartalék láb jövőbeni alkalmazások részére szabadon van hagyva

- ISO 7816-3 az elektronikus jeleket és az átviteli protokollokat írja le.
- ISO 7816-4 a belső utasítások leírását tartalmazza.
- ISO 7816-5 alkalmazások kezelésének leírását tartalmazza.
- ISO 7816-6 az adatelemeket írja le.
- ISO 7816-7 SCQL (Structured Card Query Language)

A szabvány az utóbbi időben újabb alpontokkal bővült, amelyek közül két alpont is a biztonsági követelményeket határozza meg. Az intelligens kártyák területén is eleinte a működés volt a fontos, de terjedését követően a biztonságos működésre helyeződik át a hangsúly.

Fizikai felépítés

A mai intelligens kártyák 8 bites processzorral rendelkeznek, amelyet alacsonyszintű (assembly) nyelven programozhatók. A kártyákon általában van a tartalmát megőrző és a tartalmát nem megőrző írható, olvasható memóriákkal (EEPROM, RAM) illetve csak olvasható memóriával (ROM). Ezek az erőforrások korlátozottak, de valószínűleg a hardware eszközök fejlődésével ezek a korlátok majd tágulni fognak. Az 1997 őszén Párizsban megrendezett CarteS ’97 konferencián a Gemplus által bemutatott különdíjas Cascade kártya például 32 bites RISC processzorral, 32 kilobyte nem felejtő flash memóriával és 8 kilobyte-os ROM memóriával rendelkezik. A kártya ezek segítségével saját file-rendszert tud fenntartani, amelyet a kártya operációs rendszere kezel. A kártyán nem található tápegység, a működéséhez a szükséges feszültséget az olvasó biztosítja. A kártya az olvasóval adatcsomagok segítségével kommunikál (APDU= Application Protocoll Data Unit), amelynek struktúráját az ISO 7816-4 szabvány írja le (ld. később). A kapcsolat létrehozását az olvasó kezdeményezheti, a kártya csak az olvasótól érkezett kérésekre válaszolhat (master-slave modell).

A Java alapú kártya általános leírása

A kártyás alkalmazások legnagyobb előnye a multifunkcionalitás lehetősége. Egyetlenegy kártya képes többféle információt is hordozni, és ezáltal több szolgáltatást is nyújtani. A kártyában található chip pedig képes az adatok védelmét ellátni, képes eldönteni az érkezet kérelem alapján, hogy a felhasználónak és a szolgáltató terminájának van-e joga olvasni a különböző adatokat. A rendszereket célszerű lenne úgy tervezni, hogy minden embernek van egy intelligens kártyája, amely a személyes adatokat védi, és a különböző szolgáltató cégek rendelkezésére bocsát olyan adatterületeket, amelyekkel az szabadon gazdálkodhat. Minden alkalmazás levédheti a saját adatait, amelyekhez így más szolgáltatók nem férhetnek hozzá (esetleg maga a kártyatulajdonos is csak korlátozott jogokkal rendelkezik).

A fentebb vázolt filozófia megvalósítása elképzelhetetlen a kommunikáció és a szolgáltatások szabványosítása nélkül. Ezért nagyon fontos a szabványok egyértelmű megfogalmazása.

A másik probléma a ma használt operációs rendszerek sokféleségéből adódik. Már a világháló megjelenésekor felmerült az az igény, hogy jó lenne valamilyen egységes és magas szintű programozási nyelvet kialakítani, amelynek segítségével a web-es alkalmazások könnyen elkészíthetőek. Ezt a platformfüggetlenséget a Java nyelv valósítja meg.

A kártyás fejlesztések terén is szembetalálkoztak ezzel a problémával a fejlesztők, és megpróbálták a Java nyelv logikáját ezekre a rendszerekre is alkalmazni. Elkészült a Java Card 2.0 szabvány, a tavalyi év legutolsó napján megjelent az első java alapú kártya, a Cyberflex. Azóta néhány más, kártyákat gyártó cég is megjelent a piacon a saját fejlesztésű Java kártyával, amelyek részben megfelelnek a Java Card 2.0 szabványnak. Ezek a következők: GemXpresso (Gemplus), Odyssey (Bull), C@puccino (Giesecke &Devrient), GalactIC (DeLaRue).

A Java filozófiájához hasonlóan ez a szabványos megoldás lehetővé teszi, hogy a kártyán egy alkalmazás csak a fontos adatokat tárolja, és az alkalmazás kódját minden alkalommal letöltse a megfelelő helyről (az alkalmazás helyét jelző hivatkozás természetesen a kártyán van). Így ha az alkalmazás hibájából, vagy ha az alkalmazást azóta fejlesztették, nem kell lecserélni az összes kártyát, hanem elég csak a szervergépen megváltoztatni az alkalmazás kódját.

A Java szabványyról egyébként érdemes megjegyezni, hogy az eszközös alkalmazások világából indult el, mint egy szabvány, amely megpróbálta egységesíteni az alkalmazások implementációs módszereit. A nyelv csak később vált az Internet programozási nyelvévé. A Java Card szabvány pedig tulajdonképpen visszatér az eszközös alkalmazásokhoz.

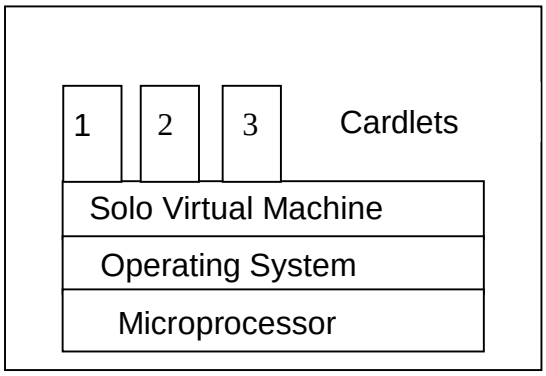
A Java kártyás alkalmazások eddigi tapasztalatai azt mutatják, hogy a Java kártya bár jelenleg még megközelítőleg harmincszor lassabb a többi, csak alacsony szinten programozható kártyáknál, mégis biztonságos, és a kártyás rendszerek egyre szélesebb körű elterjedésekor meg fognak térülni a szabványos megoldás alkalmazásának költségei.

A Java szabványhoz minden szakértő és fejlesztő nagy reményeket fűz a kártyák területén is, bár a megvalósítás nem mindig igazolja vissza az elképzeléseket. Több Java alkalmazás, és a kártyás program elkészítésének tapasztalatai után állíthatom, hogy a Java nyelv platformfüggetlensége a valóságban nem teljesen igaz! Ez szembetűnő még a két Cyberflex kártya (a Core és a fejlettebb Multi8K) egymás közötti inkompatibilitásában is.

A kártya, amelyet a Schlumberger cég honi képviselőjének jóvoltából használhattunk, alkalmas egy kártyás alkalmazás szemléltetésére. Mi az ősszel esedékes új diákigazolvány apropóján egy diákigazolványos alkalmazást készítettünk. Az elképzeléseink szerint a diákigazolványnak az addigi feladatán kívül más szolgáltatásokat is nyújtania kell. Szerintünk a diákigazolvány kiválóan alkalmas lehet sokoldalú feladatok ellátására, mint index, vizsgajelentkezés, elektronikus pénztárca, és egyebek.

A Java kártya operációs rendszere

Az alábbi ábra a Java kártya architektúráját szemlélteti:



A Java kártya operációs rendszere valósítja meg a kártya műveleteit. Ezek a műveletek a Nemzetközi Szabványügyi Hivatal intelligens kártyákra vonatkozó szabványának felel meg, amely az ISO 7816 nevet viseli.

A kártyán megtalálható egy Java virtuális gép (Solo Virtual Machine). Ez értelmezi és hatja végre a java byte-kódhoz hasonló, (és abból származó) „Solo" byte-kódot.

Erre épülnek a Java Cardlet-ek, amelyek lehetőséget nyújtanak különféle alkalmazások implementálására. Az elkészült Cardlet (cardlet.java) fordítása után kapott java byte-kódot (carlet.class) a mksolo program segítségével konvertálhatjuk „Solo" byte-kóddá (cardlet.bin). Ezt a kódot a kártya virtuális gépe (Solo Virtual Machine) már értelmezni tudja.

A kártyán található operációs rendszer szolgáltatásait a Cardlet-ek számára a _OS.java osztály teszi elérhetővé. Ennek az osztálynak a statikus, natív metódusai segítségével hajthatunk végre elemi kártyaműveleteket. A továbbiakban az operációs rendszer illetve annak műveletei alatt ezt az osztályt illetve ennek műveleteit értem.

A kártyán ezen kívül található egy alapértelmezett file-kezelő (Default loader). Ennek segítségével létrehozhatjuk a programunk futásához elengedhetetlen file-okat, illetve a programot tartalmazó file-t. Ennek segítségével tölthetjük le és futtathatjuk a programunkat.

A kártya hardware-es korlátai:

- Az alkalmazások tárolására szolgáló memóriaterület: 2.8 Kbyte
- Az alapértelmezett betöltő 1.2 Kbyte helyet foglal.
- Veremkapacitás: 32 byte
- A dinamikusan lefoglalható memóriaterület is korlátos.

A kártya software-es korlátai:

- A kártyán található virtuális gép csak három elemi típust ismer: boolean, byte, short
- A Cardlet-nek feltétlenül kell tartalmaznia egy main metódust (public static void main(String[] args)), amelyben a String[] típusú args változó nem jön létre.
- Ajánlott a Cardlet valamely pontján az Execute((short) 0, (byte) 0) sor beszúrása, amely visszaadja az alapértelmezett betöltőnek a vezérlést.

- Nem generálhatók kivételek.
- A memóriaméret korlátossága miatt az osztályszerkezetnek egyszerűnek kell lenni, minél több helyen kell konstansokat használni, illetve a már felhasznált változókat újra felhasználni.
- Ajánlott a program valamelyik pontjára az Execute ((short) 0, (byte) 0) sor beszúrása, amely visszaadja a vezérlést az alapértelmezett betöltőnek.

A kártya file-rendszere

A kártyán minden file egy 16 byte-os fejléccel kezdődik. Ez tartalmazza a file-val kapcsolatos legfontosabb információkat. A felépítése a következő:

Byte	0	1	2	3	4	5	6	7
File adatai	File hossz		File azonosító		Típus	Státusz	Rek. h.	Rek. sz.
ACL	Default	CHV1	CHV2	AUT0	AUT1	AUT2	AUT3	SUP

A file hossza tartalmazza a fejléc hosszát is, ezért legalább 16 byte kell legyen.
A file típusa alapján a 4. byte lehet:

0x20	könyvtár
0x02	bináris file
0x03	program file
0x1D	ciklikus file
0x19	változó rekord file
0x0C	fix rekord file

A fejléc utolsó nyolc byte-ja (Access Control List) a különböző jogosultságokat tartalmazza. A kártya kétféle azonosítási módszert alkalmaz, és azokon belül kettő (CHV1, CHV2) illetve négy (AUT0 - AUT3) identitás létezik. A megfelelő byte-ok a megfelelő identitásokra vonatkozó jogosultságokat írják le. A jogosultsági rendszer részétesebb leírását lásd később.

Alapértelmezésben a kártya file-rendszere két file-t tartalmaz:

0x3F00	a gyökérkönyvtár
0x0011	az identifikációs kódokat tartalmazza, és a főkönyvtárban található. A szerepéről később részletesebben is szó lesz

A kártya biztonsági rendszere

A kártyán tárolt adatok többszörös védelem alatt állnak. A hardware-es szinten lehetetlen az adatok dekódolása, mert a kártya fizikai megvalósításakor figyeltek rá, hogy az adatokat ne lehessen a software-es biztonsági rendszer megkerülésével, valamilyen elemi fizikai úton leolvasni. A software-es védelem négyjegyű PIN kód alkalmazására illetve ennél hosszabb, kulcsok alkalmazására épül.

A kártya a nyolc identitást különböztet meg, ami azt jelenti, hogy a különböző műveletek végrehajtási jogai alapján a kéréseket intéző felhasználók nyolc csoportba oszthatóak (a UNIX rendszereknél ilyen szempontból egy file megváltoztatásakor megkülönböztetjük a file

tulajdonosát (owner), a tulajdonossal egy csoportba tartozó felhasználókat (group) és a többi felhasználót (other)).

A kártya esetén létezik egy alapértelmezett felhasználó (default), ami semmilyen különleges joggal nem rendelkezik, majd létezik a kártyának tulajdonosa, társtulajdonosa, és néhány szolgáltató, aki a saját adatait mások számára hozzáférhetetlenül tárolhatja a kártyán. A nyolcadik (SUP=supervisor) identitás segítségével, egy alkalmazás saját konfigurációs file-okat tarthat fenn, amelyek külső felhasználó számára rejtettek. Nagyjából ez a logika vezérelte a rendszer kitalálóit.

A tulajdonosokat (CHV1, CHV2) 4 jegyű PIN kód segítségével azonosítják. (CHV = Card Holder Verification). A másik azonosítási módszer (AUT0 – AUT3) egy tetszőleges hosszúságú kulcs alapján történik.

A kártya kezdetben az alapértelmezett felhasználó jogait veszi figyelembe, majd kérésre, a PIN kód illetve a kulcs ellenében megadja a jogokat a megfelelő műveletek végrehajtására. Így az identitások egymásra tevődhetnek, és a felhasználó megkapja mindazokat a jogokat, amelyek valamelyik ellenőrzött identitásához tartoznak. A kártya kikapcsolásakor vagy újraindításakor ismét csak az alapértelmezett jogokkal rendelkezik majd a felhasználó.

Az előbbiekben szereplő, az operációs rendszerekre jellemző, és az itt használt identitással analóg „felhasználó” kifejezés takarhat valódi felhasználót, aki beüti a PIN kódját a terminál mellett, de itt inkább egy alkalmazásra, futó programra gondolok, amely futása közben kérhet jogokat, és lekérdezhet, illetve módosíthat adatokat.

A kártya tehát kétféle azonosítási módszert alkalmaz:

PIN kód alapján történő azonosítás

A PIN kód ellenőrzését, (és ezzel a jogok iránti kérelmet) a VerifyCHV parancs segítségével lehet kérni. A parancs végrehajtásakor az operációs rendszer megkeresi, először az aktuális, majd az azt tartalmazó könyvtárak-ban 0x0000 (CHV1) illetve a 0x0100 (CHV2) file-t, ami a PIN kódra vonatkozó információkat tartalmazza. Ha a megadott PIN kód helytelen, egyel csökken a hátralévő próbálkozások számát tartalmazó mező. Ha a PIN kód helyes, a lehetséges próbálkozások számát jelző mező visszaáll a kezdeti értékére, és az alkalmazás megkapja az illető PIN kódhoz kapcsolt jogokat. A PIN kódot inaktívra állíthatjuk, illetve módosíthatjuk, a ModifyCHV parancs segítségével. (Ez a művelet is egy PIN kódhoz van kötve). Mivel az operációs rendszer először az aktuális könyvtárban keresi a PIN kódot leíró file-t, minden alkalmazás saját PIN kóddal rendelkezhet, ha abban a könyvtárban, amelyet az alkalmazás használ, fenntart egy (PIN kódot leíró) CHV file-t.

A PIN kódot leíró file szerkezete a következő:

Eltolás	***Tartalom***
0	Aktivációs mutató (01 = Aktív)
1-2	RFU- kódolási eljárás
3-12	A PIN kód értéke
11	A megengedhető sikertelen próbálkozások száma
12	A hátralévő lehetőségek száma
13-20	A deaktiválás kódja
21	Megengedett deaktiválási próbálkozások száma
22	Hátralévő lehetőségek száma

Kulcs alapján történő azonosítás

A kulccsal történő azonosítás alkalmával az AUT0 – AUT3 identitást ellenőrzi az operációs rendszer. A kulcsokra vonatkozó adatok a 0x0011 file-ban, és szükségszerűen a főkönyvtárban találhatóak. A kulcs ellenőrzését a VerifyKey parancs segítségével lehet kezdeményezni.

A kulcsokat leíró file szerkezete:

Eltolás	Tartalom
0	RFU – kódolás
1	Kulcs hossza (X)
2	Key tag ???
3-3+(X-1)	A kulcs értéke
3+X	Lehetséges próbálkozások száma
3+X+1	Hátralévő próbálkozások száma
3+X+2	Következő kulcs hossza
3+X+3	Key tag ???
3+X+4-3+...	Következő kulcs értéke

File-ra vonatkozó jogosultsági rendszer (Access Control List)

A file-okra vonatkozó különböző jogokat minden file fejlécének utolsó nyolc byte-ja tartalmazza (ACL). Ez a rész határozza meg, hogy a különböző file műveleteket milyen identitás birtokában kezdeményezhet a felhasználó.

A lehetséges műveletek a következők:

	Byte érték	**Elemi file**	**Rekord file**	**Könyvtár**
0. bit	0x01	Olvasás	Olvasás	Listázás
1. bit	0x02	Írás	Írás	Törlés
2. bit	0x04	Futtatás	Rekord létrehozás	ACL megváltoztatása
3. bit	0x08	Érvénytelenítés	Érvénytelenítés	Érvénytelenítés
4. bit	0x10	Visszaállítás	Visszaállítás	Visszaállítás
5. bit	0x20	Egyéb 0	Egyéb 0	File létrehozás
6. bit	0x40	Egyéb 1	Egyéb 1	Egyéb 1
7. bit	0x80	Egyéb 2	Egyéb 2	Egyéb 2

A táblázatban az egyébbel jelzett jogok alkalmazásfüggők lehetnek.

A jogosultság táblázat (ACL) felépítése:

0. byte	1. byte	2. byte	3. byte	4. byte	5. byte	6. byte	7. byte
Default	CHV1	CHV2	AUT0	AUT1	AUT2	AUT3	SUP

A táblázatban minden bejegyzés egy byte-ot jelöl, amely az illető identitásra vonatkozó jogokat tartalmazza. A byte minden bitje egy műveletre vonatkozó jogosultságot jelzi (lásd fenti táblázat). Az illető identitásnak van joga a művelet végrehajtására, ha a bit 1, és nincs rá joga, ha a bit értéke 0.

A fenti táblázat második oszlopa byte értékét jelzi, ha van jog a műveletre. (Ha több jogot is be akarunk állítani egy identitás esetén, a táblázat második oszlopában szereplő értékeket össze kell adni).

Például, ha csak a CHV2 identitásnak van írásjoga, és minden egyéb műveletet akárki végrehajthat egy bináris file-on, akkor az ACL a következő értéket tartalmazza:

FD 00 02 00 00 00 00 00 (01+04+08+10+20+40+80=FD)

A kártya operációs rendszerének eljárásai négy fő feladatot látnak el:

Kommunikációs függvények

Ezen függvények segítségével valósítható meg a kapcsolat a külvilággal. A kapcsolat az ISO 7816 szabvány szerint történik.

A kártya üzeneteket fogadhat az olvasótól, majd ezekre válaszolhat. A kártya maga nem kezdeményezhet kapcsolatot, csak a kártyaolvasó kéréseire reagálhat.

A kártya az ISO szabvány T=0 protokollal fogadja az üzenetet, amely egy aszinkron, „half-duplex" byte átviteli protokoll. A kommunikáció során a kártya az érkező byte-okat egy ideiglenes tárolóban helyezi el, majd ha a tároló a megfelelő hosszúságban feltöltődött adatokkal, átveszi az üzenetet.

Egy ISO kérés két részből áll:

- Parancsrészből, amely az elvégzendő műveletet, és annak néhány paraméterét tartalmazza.

  A parancs 5 byte hosszú:

Osztály	Utasítás	1. Paraméter	2. Paraméter	Adat hossza
CLA	INS	P1	P2	LNG

  - Az első két mező, az osztály (CLA) ill. az utasítás (INS) mező határozza meg, hogy milyen műveletet kell a kártyának végrehajtani.
  - A paraméterek módosítják a kiválasztott művelet hatását.
  - Az 5. byte határozza meg a parancsrészt követő adatrész hosszát.
- Adatrészből, amely a művelethez szükséges adatokat tartalmazza. Az adatrész hosszát a parancsrész utolsó paramétere határozza meg.

A parancsokra adott válaszokból ki kell derülnie, hogy sikeres volt-e a művelet, és amennyiben nem, valamilyen módon a hiba okát jelezni kell a külvilág felé. Ezért a kártya a kért művelet végrehajtása után mindig küld két ellenőrző byte-ot. Ez a hibakód egy egybyte-os belső hibakódból származik. A konverziót a SendStatus függvény hajtja végre.

A hibakódok jelentéseit a következő táblázat tartalmazza:

Konstans	***Érték***	***Státusz***
ST.SUCCESS	*0x00*	*9000*
ST.INVALID_P3	*0x60*	*6700*
ST.FAILURE	*0x20*	*6200*
ST.INVALID_CRC	*0x21*	*6280*
ST.INVALIDATE	*0x22*	*6281*
ST.NO_ACTION	*0x30*	*6300*
ST.INVALID_CHV	*0x30*	*6381*
ST.INVALID_KEY	*0x30*	*6381*
ST.UNSTABLE	*0x50*	*6500*
ST.WRITE_ERROR	*0x52*	*6581*
ST.INTERPRETER_ERROR	*0x54*	*6583*
ST.NO_KEY_DEFINED	*0x82*	*6981*
ST.NO_CHV_DEFINED	*0x82*	*6981*
ST.NO_ACCESS	*0x83*	*6982*
ST.CHV_LOCKED	*0x84*	*6983*
ST.KEY_LOCKED	*0x84*	*6983*
ST.DIR_NOT_EMPTY	*0x85*	*6984*
ST.LOADEXE_LOCKED	*0x86*	*6985*
ST.NOT_LAST	*0x98*	*6A00*
ST.UNKNOWN	*0x90*	*6A00*
ST.INVALID_ID	*0x90*	*6A00*
ST.INVALID_FILE_TYPE	*0x91*	*6A80*
ST.NOT_FOUND	*0x93*	*6A82*
ST.INVALID_RECORD_NB	*0x94*	*6A83*
ST.INVALID_FILE_SIZE	*0x95*	*6A84*
ST.OUT_OF_FILE	*0x96*	*6A85*
ST.OUT_OF_RECORD	*0x96*	*6A85*
ST.LIMIT_REACHED	*0x97*	*6A86*
ST.INVALID_PARAMETER	*0xA0*	*6B00*
ST.INVALID_LENGTH	*0xA1*	*6B80*
ST.INVALID_SPEED	*0xA2*	*6B81*
ST.INVALID_MODE	*0xA3*	*6B82*
ST.INS_NOT_SUPPORTED	*0xB0*	*6D00*
ST.INVALID_CLASS	*0xC0*	*6E00*

A konverzió a következő táblázat alapján történik:

A két byte első byte-ja az első négy bit-ből, a második byte-ja pedig az utolsó négy bit-ből alakul ki:

Az egy byte-os hibakód négy bit-je	Első négy bit esetén	Utolsó négy bit esetén
0	0x90	0x00
1	0x61	0x80
2	0x62	0x81
3	0x63	0x82
4	0x64	0x83
5	0x65	0x84
6	0x67	0x85
7	0x68	0x86
8	0x69	0x87
9	0x6A	0x50
0xA	0x6B	0x10
0xB	0x6D	0x18
0xC	0x6E	0x00
0xD	0x6F	0x00
0xE	0x98	0x00
0xF	0x00	0x00

Például a 0xAB egy byte-os hibakód első négy bit-je (0xA) a táblázat második oszlopa alapján 0x6B byte-ot indukál, míg az utolsó négy bit (0xB) a táblázat harmadik oszlopa alapján 0x18-at ad. Így a kapott két byte-os hibakód a 0x6B18 lesz.

A kártyán található java virtuális gép (JVM) által generált kivételek alkalmával a kártya újraindul (Reset). Ebben az esetben a következő hibakódok jelzik a hibát:

Kivétel	***Érték***	***Státusz***
OutOfMemoryException	*0xF1*	*0080*
IllegalOpcodeException	*0xF2*	*0081*
IllegalTypeException	*0xF3*	*0082*
ArrayIndexOutOfBoundsException	*0xF4*	*0083*
NullPointerException	*0xF5*	*0084*
ArithmeticException	*0xF6*	*0085*
NoSuchMethodException	*0xF7*	*0086*
ClassCastException	*0xF8*	*0087*
NoSuchFieldException	*0xF9*	*0050*
StackOutOfRangeException	*0xFA*	*0010*
LocalOutOfRangeException	*0xFB*	*0018*
AthrowException	*0xFC*	*0000*
NoSuchClassException	*0xFD*	*0000*
IncompatibleFieldException	*0xFE*	*0000*
CallTooDeepException	*0xFF*	*0000*
NegativeArrayExcetpion	*0xD1*	*6F80*

A legfontosabb kommunikációs eljárások:

```
static native byte GetMessage (byte buffer[], byte expected
    length, byte ack_code);
```

Leírás: ISO T=0 protokollal átvesz egy üzenetet.

Paraméterek:

buffer	ide fognak kerülni a beolvasott byte-ok
expected_length	beolvasandó byte-ok száma
ack_code	- 00, ha új utasítást olvasunk - INS, ha egy már beolvasott utasításhoz tartozó adatot akarunk beolvasni - ~INS, ha a byte-okat egyenként (slave módban) szeretnénk beolvasni

Visszatérített érték: ST.SUCCESS (0x9000)

```
static native byte SendMessage (byte buffer[], byte
    data_length);
```

Leírás: ISO T=0 protokollal elküld egy üzenetet.

Paraméterek:	buffer	ennek a tartalmat küldi el
	data_length	az elküldendő byte-ok száma
Visszatérített érték:	ST.SUCCESS	az üzenetet sikeresen elküldte
	Más	az üzenet elküldése közben hiba lépett fel

```
static native void SendStatus (byte status);
```

Leírás: konvertálja és elküldi az állapotjelző byte-ot. (ls korábban)

Paraméterek:	status	az egy byte-os belső hibakód.
Visszatérített érték:	nincs	

File-kezelő rutinok

A file-kezelő eljárások a belső file-rendszer kezelését valósítják meg.

Minden file egy fejléccel kezdődik, amely tartalmazza a file tulajdonságait. Ez 16 byte hosszú. A file hossza tartalmazza a fejléc hosszát is, így legalább 16 byte kell legyen.

A file-ok a memóriában egymást követik, és a memória feldarabolásának elkerülése végett ezt a struktúrát a kártya fenn is tartja. Ez azt jelenti, hogy más file-ok közé ékelt file-t nem lehet törölni. Így a mindig csak az utoljára létrehozott file törölhető, azaz a file-ok létrehozásának és törlésének rendszere verem jellegű (last in first out).

```
static native byte CreateFile (byte file_hdr[]);
```

Leírás: Létrehozza a megfelelő fejléccel rendelkező file-ot.

Paraméterek:	file_hdr[]	A 16 byte-os fejléc.
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.OUT_OF_FILE	nincs elég memória.
	ST.INVALID_ID	a file már létezik.
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.WRITE_ERROR	írás közben hiba történt, nem jött létre a file.
	ST.INVALID_FILE_SIZE	A file mérete kisebb mint 24 byte.
	ST.INVALID_PARAMETER	A rekordok hossza túl nagy.

```
static native byte DeleteFile (short file_Id);
```

Leírás:	Törli a megadott file-ot.	
Paraméterek:	file_Id	a file azonosítója.

Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NOT_LAST	nem az utolsó file (ld.: korábban)
	ST.DIR_NOT_EMPTY	a könyvtár nem üres
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.NOT_FOUND	nem létezik ilyen file az aktuális könyvtárban .
	ST.INVALID_ID	nem létezik ilyen azonosító
	ST.WRITE_ERROR	az memóriába írásnál hiba történt, esetleg sikertelen törlés.

```
static native byte GetFileInfo (byte file_hdr[]);
```

Leírás:	Megadja az aktuális file fejlécét	
Paraméterek:	file_Id	ide kerül a file fejléce
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.UNKNOWN	ismeretlen hiba

```
static native byte  ReadBinaryFile (short offset, byte
    data_length, byte buffer[]);
```

Leírás:	Olvas az aktuális fileból	
Paraméterek:	offset	a file-nak ettől a pozíciójától kezdődik az olvasás.
	data_length	a beolvasásra kerülő byte-ok száma
	buffer	ide kerül a file tartalma
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.INVALID_TYPE	az aktuális file nem bináris
	ST.OUT_OF_FILE	…

```
static native byte SelectFile (short file_Id);
```

Leírás:	Beállítja az aktuális file-t	
Paraméterek:	file_Id	a kiválasztásra kerülő file azonosítója
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NOT_FOUND	nem létezik ilyen file
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.

```
static native byte WriteBinaryFile(short offset,
    bytedata_length, byte buffer[]);
```

Leírás:	Írás az aktuális file-ba	
Paraméterek:	offset	ettől a pozíciótól kezdve írjuk a byte-okat
	data_length	kiírandó byte-ok száma
	buffer	ezek a byte-ok kerülnek kiírásra
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.INVALID_TYPE	az aktuális file nem bináris
	ST.OUT_OF_FILE	a parancs a file keretein kívülre szól
	ST.WRITE_ERROR	hiba az írás közben

Biztonsági metódusok

```
static native byte CheckAccess (byte ac_action);
```

Leírás:	Ellenőrzi, hogy a művelethez az aktuális identitásnak meg van-e a megfelelő jogosultsága	
Paraméterek:	ac_action	AC_READ (0) AC_WRITE (1) AC_CREATE_RECORD (2) AC_CREATE_FILE (15) AC_DELETE_FILE (11) AC_CHANGE_ACL (12) AC_SEEK (0) AC_EXECUTE (2) AC_CUSTOM_1 (5) AC_CUSTOM_2 (6) AC_CUSTOM_3 (7)
Visszatérített érték:	ST.ACCESS_CLEARED	hozzáférés engedélyezve
	ST.ACCESS_DENIED	nincs meg a megfelelő jogosultság.
	ST.INVALIDATE	a file érvénytelenítve van

```
static native byte GetFileACL (byte file_acl[]);
```

Leírás:	Megadja az aktuális file jogosultsági táblázatát (ACL)	
Paraméterek:	file_acl	ide kerül a file jogosultsági táblázata

Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.INVALID_TYPE	nincs kiválasztott file

```
static native byte GetFileStatus();
```

Leírás: Megadja az aktuális file státuszát (fejléc 5. Byte-ja)

Paraméterek: nincs

Visszatérített érték:	n	a file státusza

```
static native byte GrantSupervisorMode();
```

Leírás: Kéri a SUP identitás azonosítását (nincs kulcs)

```
static native void LogOutAllId();
```

Leírás: Visszaállítja az alapértelmezett jogosultságokat

```
static native byte ModifyCHV(byte CHV_number, byte
    old_CHV_buffer[], byte new_CHV_buffer[], byte
    unblock_flag);
```

Leírás: Megváltoztatja a PIN kódot. Nem változik meg az identitás (ha eddig nem volt leellenőrizve a megfelelő identitás, megváltozik a PIN kód, de nem kapja meg az identitást a felhasználó.

Paraméterek:	CHV_number	ennek az identitásnak megfelelő PIN kód változik meg (CHV1 vagy CHV2).
	unblock_flag	0→ az identitást azonosító PIN kód változik meg 1→ az PIN kódot érvénytelenítő PIN kód változik meg
	old_CHV_Buffer	régi PIN kódot tartalmazó tömb
	new_CHV_Buffer	új PIN kódot tartalmazó tömb
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.INVALID_CHV	érvénytelen identitás
	ST.CHV_LOCKED	a PIN kód érvénytelenítve van

```
static native byte RevokeSupervisorMode();
```

Leírás: visszaadja a SUP identitáshoz tartozó jogokat

```
static native byte SetFileACL file_acl[]);
```

Leírás:	beállítja az aktuális file jogosultsági táblázatát (ACL). A művelethez szükséges a „ACL megváltoztatása ” jog az aktuális könyvtárra.	
Paraméterek:	file_acl	új jogosultsági táblázat
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.INVALID_TYPE	nincs kiválasztott file

```
static native byte SetFileStatus (byte file_status);
```

Leírás:	beállítja az aktuális file státuszát	
Paraméterek:	file_status	az új státusz
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság.
	ST.WRITE_ERROR	hiba az írásnál, esetleg sikertelen művelet
	ST.UNSTABLE	a file bizonytalan státuszban van

```
static native byte VerifyCHV (byte CHV_number, byte
    CHV_buffer[], byte unblock_flag);
```

Leírás:	Ellenőrzi a megadott PIN kódot. Megadott számú rossz próbálkozás után a PIN kód érvénytelené válik, és ezt lehet feloldani a második PIN kód segítségével.	
Paraméterek:	CHV_number	az ellenőrzésre kerülő identitás
	CHV_buffer	a PIN kódot tartalmazó tömb
	unblock_flag	0→ a PIN kódot ellenőrizzük 1→ az érvénytelenítő PIN kódot ellenőrizzük.
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.INVALID_CHV	érvénytelen identitás
	ST.CHV_LOCKED	érvénytelen PIN kód

```
static native byte  VerifyKey (byte key_number, byte
    key_buffer[], byte key_length);
```

Leírás:	Ellenőrzi az adott identitásnak megfelelő kulcsot. Megadott számú sikertelen próbálkozás után a kulcs érvénytelenné válik, amit nem lehet feloldani.	

Paraméterek:	key_number	az identitást azonosító szám
	key_buffer	a kulcsot tartalmazó tömb
	key_length	a kulcs hossza
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.INVALID_KEY	rossz kulcs
	ST.KEY_LOCKED	érvénytelen kulcs
	ST.NOT_FOUND	a kulcsokat tartalmazó file nem létezik

Egyéb segédfüggvények

- AvailableMemory
- CompareBuffer
- ComputeXor
- CopyBuffer
- Execute
- GetApplicationId
- GetFileName
- GetFileSize/GetFileLength
- GetFileType
- GetIdentity
- GetRecordLength
- GetRecordNb
- ResetCard
- SendATR
- SetDefaultATR

```
static native byte Execute (short fileId, byte flag);
```

Leírás:	Futtatja a kijelölt programot.	
Paraméterek:	fileId	a programot tartalmazó file azonosítója
	flag	a program végrehajtásának módját határozza meg. 7. bit egyes→a program azonnal elindul, majd az első újraindítás alkalmával ismét az alapértelmezett betöltő indul el. 3. bit egyes→a program a következő újraindítás alkalmával indul csak el, de az azt követő újraindításkor nem adódik vissza a vezérlés az alapértelmezett

		betöltőnek. Erről az illető programnak kell gondoskodni. 0x00 →bármi történjen a következő indításkor az alapértelmezett betöltő fut majd.
Visszatérített érték:	ST.SUCCESS	sikeres parancs
	ST.INVALID_FILE_TYPE	a megjelölt file nem program file
	ST.NO_ACCESS	nincs meg a megfelelő jogosultság
	ST.NOT_FOUND	a megjelölt file nem létezik

A fejlesztésben rendelkezésre álló software-k

JDK1.1.5

A fejlesztés során a JDK 1.1.5 fejlesztőkörnyezetet használtam.

A Reader osztály

A Reader osztály valósítja meg a kapcsolatot a terminál és a kártya között. Ennek az osztálynak a segítségével lehet egy Java programból üzeneteket küldeni a kártyának, illetve fogadni a válaszokat. A Reader osztály a kapcsolatot a kártyával natív metódusok segítségével valósítja meg. Ezek a natív metódusok Windows-os környezetre vannak implementálva, ezért ezt az osztályt csak Windows alatt lehet használni. Amennyiben az osztály natív metódusait más operációs rendszereken is megvalósítják majd, ez az osztály és az osztály funkcióit használó programok átültethetőek lesznek más operációs rendszerekre is. (Az alternatív operációs rendszeren természetesen léteznie kell valamilyen Java futtató környezetnek is ezen az osztályon kívül.)

A Reader osztály legfontosabb műveletei a következők:

ReaderAllocate(String ReaderName, String PortName)

Leírás:	Ez a parancs felépíti a kapcsolatot az olvasóval.	
Paraméterek:	ReaderName	Az olvasó típusa, esetünkben „Reflex60”. Ezenkívül lehet még Reflex20, SCR60, SCT, UCRS-1, UCRS-3
	PortName	Ez a paraméter jelzi, hogy melyik port-ra van az olvasó csatlakoztatva. Ez lehet: COM1-COM4, és PCMCIA.

ReaderFree()

Leírás:	Felszabadítja az olvasót.

ReaderPowerUp()

Leírás:	A kártyaolvasó bekapcsolását kezdeményezi.

ReaderReset(StringBuffer ATR)

Leírás:	Ez a parancs felépíti a kapcsolatot az olvasóval.	
Paraméterek:	ATR	A következő induláskor ez az üzenet érkezik majd a kártyáról.

ReaderSendIsoInT0(String Command, StringBuffer Response, StatusWord SW)
ReaderSendIsoOutT0(String Command, StringBuffer Response, StatusWord SW)

Leírás:	Elküld egy üzenetet, megvárja a választ, majd az beírja a „Response” változóba.	
Paraméterek:	Command	Az elküldendő byte-okat tartalmazó karaktersorozat.
	Response	Ide kerül a kártyáról visszaküldött válasz
	SW	A státuszt jelző byte-ok értéke.

A Reader osztály ezek mellett más hasznos metódusokat is tartalmaz.

Mksolo

Ennek a programnak a segítségével konvertálható át a Java byte-kód olyan bináris állománnyá, amelyet már a kártya virtuális gépe is értelmezni tud.
A mksolo (Make Solo) programnak szüksége van olyan információkra is, amelyeket a Java fordító alapértelmezésben nem ír bele a byte-kódba, ezért az mksolo használatakor kötelező a Java fordító –g opcióját beállítani. (javac –g pin.java).

A mksolo program –o opciójával adható meg a kimeneti file neve, és a –s opció segítségével nyerhetünk részletesebb információkat az elkészült bináris állományról (mérete, várható verem ill. memóriaigénye).

Az alapértelmezett betöltő

A kártyán található egy alapértelmezett betöltő program. Ennek segítségével lehet új alkalmazásokat tölteni a kártyára, és ennek segítségével lehet a már meglévő alkalmazások valamelyikének átadni a vezérlést. Az alapértelmezett betöltő az operációs rendszer szolgáltatásainak egy részét közvetlenül is elérhetővé teszi.

Az alapértelmezett betöltő funkciói a következők:

File létrehozása

CLA	***INS***	***P1***	***P2***	***LNG***
F0	E0	0	0	0x10

Az adatrész 16 byte-ja a file fejlécét tartalmazza (ld.: korábban).

A lehetséges válaszok:

Státusz	*Leírás*
6A86	Az LNG mező helytelen (!=0x10)
6A85	Nincs elég hely a file létrehozásához
6A81	A file már létezik
6A84	A file mérete kisebb a fejléc méreténél
6A86	Az érkezett adatsor hossza nem megfelelő
6B00	A Rekordok mérete nem összeegyeztethető a file méretével
6581	Hiba lépett fel írás közben
6982	Nincs meg e megfelelő jogosultság

File törlése

CLA	*INS*	*P1*	*P2*	*LNG*
C0	E4	0	0	02

Az adatmező a file azonosítóját tartalmazza. (Pl. az 1234 file törlése: C0 E4 00 00 02 12 34)

Státusz	*Leírás*
6A50	A könyvtár nem üres
6A87	Nem ez az utolsó file
6A82	Nem létezik a file
6A81	Nem lehet kitörölni a gyökérkönyvtárat
6B00	Helytelen LNG paraméter (!=2)
6982	Nincs rá jog
9000	Sikeres parancs

Egy file kiválasztása

CLA	*INS*	*P1*	*P2*	*P3*
C0	A4	0	0	02

Az adatmező a kiválasztandó file azonosítóját tartalmazza. (Pl. 1234 file kiválasztása: C0 A4 00 00 02 12 34)

Státusz	***Leírás***
6A82	Nem létezik ilyen file
6B00	Helytelen LNG mező (!=2)
9000	Sikeres végrehajtás

Könyvtár tartalmának lekérdezése

CLA	***INS***	***P1***	***P2***	***LNG***
C0	A8	00	Fileok száma	Hossz

Az adatmező üres, a P2 paraméter jelzi, hogy a könyvtár melyik állományának a fejlécére kíváncsi a felhasználó. A P2 a keresett file sorszáma a létrehozási sorrend szerint, nulla jelzi az aktuális könyvtárat.

A kártya a parancs sikeres végrehajtása estén elküldi a megjelölt file fejlécének byte-jait, a kívánt hosszúságban.

Státusz	***Leírás***
6A85	Helytelen P2, ilyen file nem létezik
6700	Helytelen hossz
6982	Nincs meg a megfelelő jogosultság
9000	Sikeres végrehajtás

Olvasás file-ból

CLA	***INS***	***P1***	***P2***	***P3***
F0	B0	Eltolás1	Eltolás2	Length

A P1 és P2 mezők jelzik, hogy az aktuálisan kiválasztott file-ból melyik pozíciótól kezdődjön az olvasás. Az LNG mező tartalma határozza meg, hogy hány byte-ot kell kiolvasni.

Státusz	***Leírás***
6A80	Az aktuális file nem bináris
6A85	A jelzett hossz és eltolás a file-on kívülre mutat
6982	Nincs meg a megfelelő jogosultság
9000	Sikeres végrehajtás

File-ba írás

CLA	***INS***	***P1***	***P2***	***LNG***
F0	D6	Eltol.1	Eltol.2	Hossz

A P1 ill. P2 paraméterek határozzák meg azt a pozíciót, ahova az adatokat kell írni. A LNG mező határozza meg, hogy hány byte-ot kell kiírni. A beírandó byte-okat az adatmező tartalmaza.

Státusz	**Leírás**
6A80	Az aktuálisan kiválasztott file nem bináris
6A84	A jelzet eltolás és a hossz a file-on kívülre mutat
6982	Nincs meg a megfelelő jogosultság
9000	Sikeres végrehajtás

PIN kód átállítása

CLA	***INS***	***P1***	***P2***	***LNG***
A0	24	00	Szám	Hossz

A P2 paraméter határozza meg, hogy melyik (CHV1 vagy CHV2) identitáshoz tartozó PIN kódot kell megváltoztatni. Az adatmező tartalma: régi kód + új kód, mindkettő 8 byte hosszú.

Státusz	**Leírás**
6B00	Érvénytelen hossz (!=16) Helytelen P2 paraméter
6381	Helytelen a régi PIN kód
6581	Az írás közben hiba lépett fel
6981	Nincs beállítva PIN kód
6983	A PIN kód zárolva van
9000	Sikeres végrehajtás

Kulcs ellenőrzése

CLA	***INS***	***P1***	***P2***	***P3***
F0	2A	0	Nb	Length

A P2 paraméter jelzi, hogy melyik identitást (AUT0- AUT3) kell ellenőrizni. Az adatmező tartalmazza a kulcsot.

Státusz	***Leírás***
6300	Hibás kulcs
6300	Nincs ilyen sorszámú kulcs definiálva
6983	A kulcs zárolva van
9000	Sikeres végrehajtás

Program elindítása

CLA	***INS***	***P1***	***P2***	***P3***
F0	F2	Id 1	Id 2	1

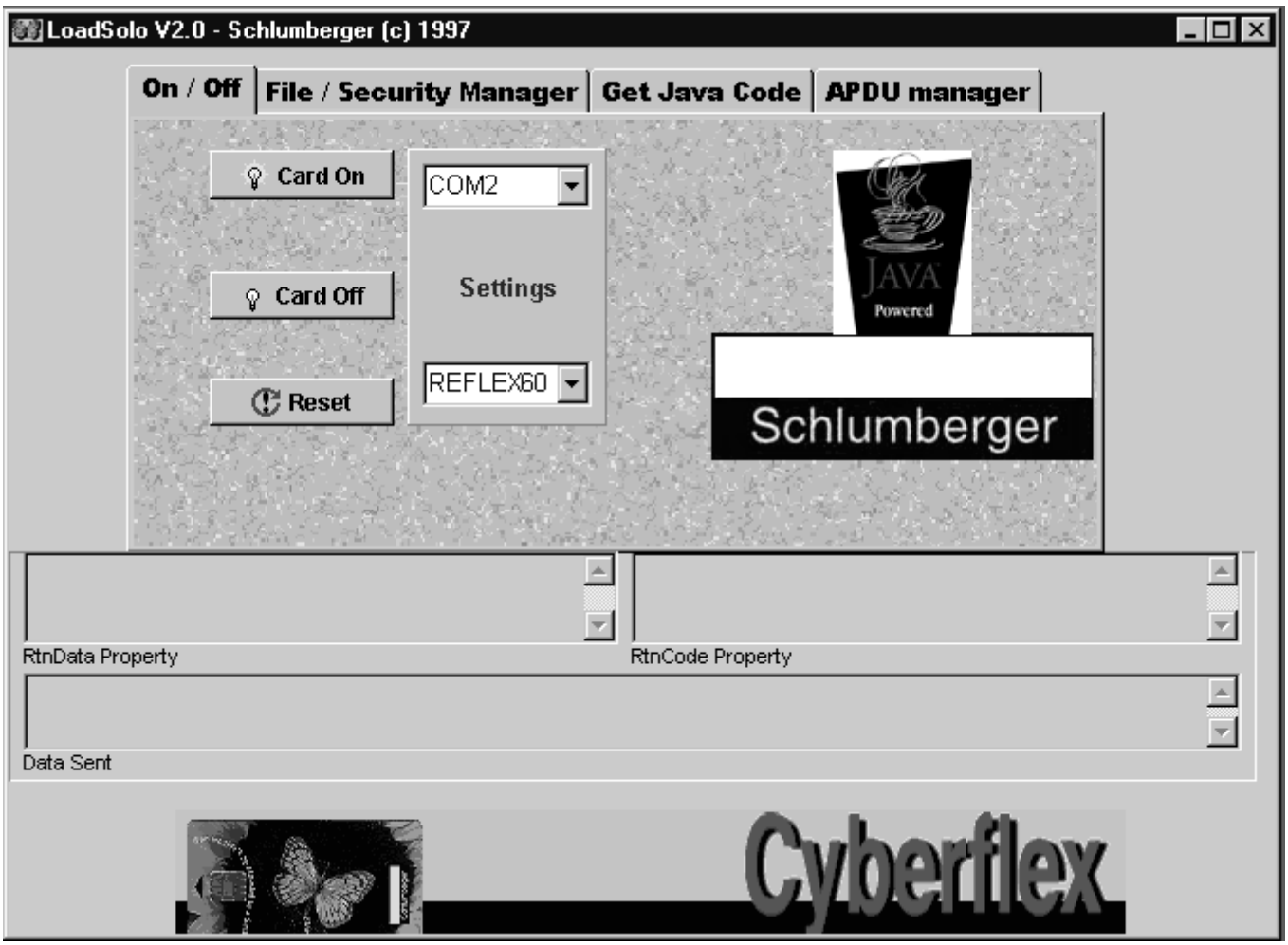
A P1 és P2 paraméterek határozzák meg az elindítandó file azonosítóját. Az adatmező egyetlen byte-ja határozza meg, hogy milyen módon adódjék át a vezérlés a Cardlet-nek (ld. korábban).

(Például ha az 1234 file tartalmazza a programot és hogy a következő indításnál adódjék át a vezérlés: F0 F2 12 34 01 0F)

Státusz	***Leírás***
6A82	Nincs ilyen file
6A80	Nem program file
6300	A file nem tartalmaz programot
6982	Nincs meg a megfelelő jogosultság
9000	Sikeres végrehajtás

Reflex Reader

A Reflex Reader egy windows alá írt kommunikációs program, amelyből kényelmes, felhasználóbarát felület segítségével különböző parancsokat küldhetünk a kártyának.

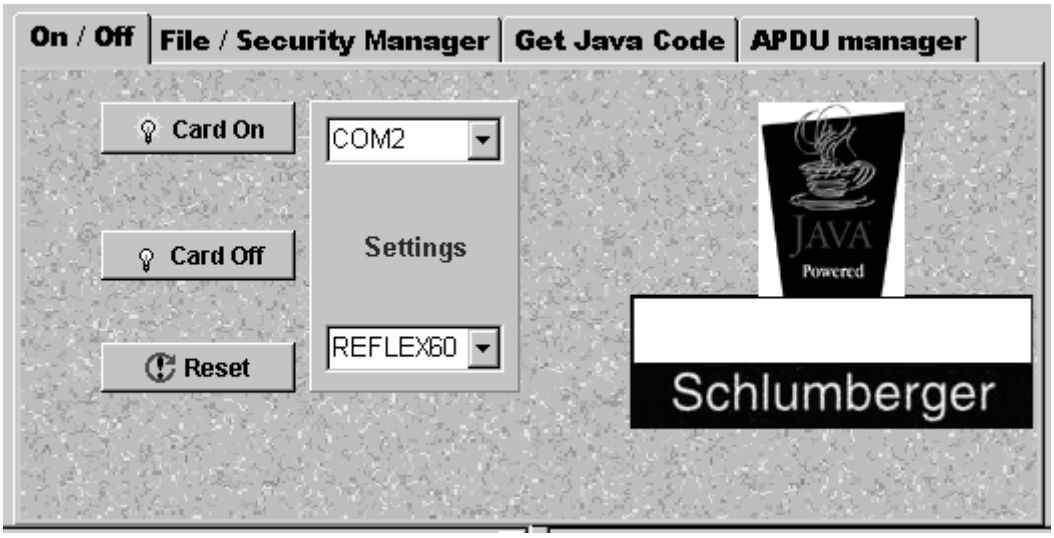


A programablak három adatmegjelenítő mezőt tartalmaz:

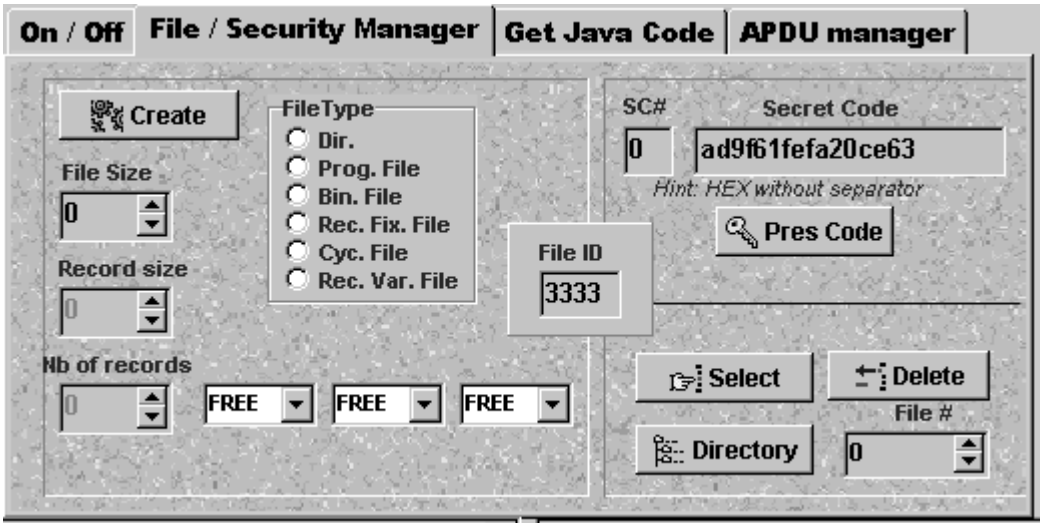
- A jobboldali, felső mezőben jelennek meg a kártyáról érkezett adatok.
- A jobboldali, alsó mezőben jelennek meg azok az adatok, amelyeket a terminál küldött a kártya felé
- A baloldali szövegmező pedig minden parancs végrehajtása utáni státusz értékét tartalmazza.

A programablak felső részében a funkciók négy csoportba vannak osztva:

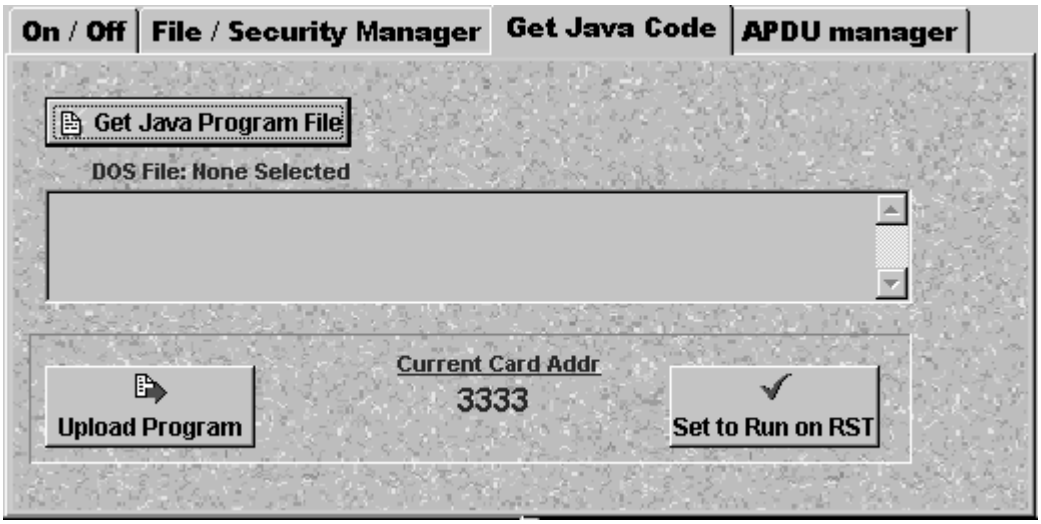
1. Az első panel segítségével az olvasó be ill. kikapcsolását tudjuk kezdeményezni. Itt található még néhány beviteli mező, ahol a port és az olvasó típusát állíthatjuk be.



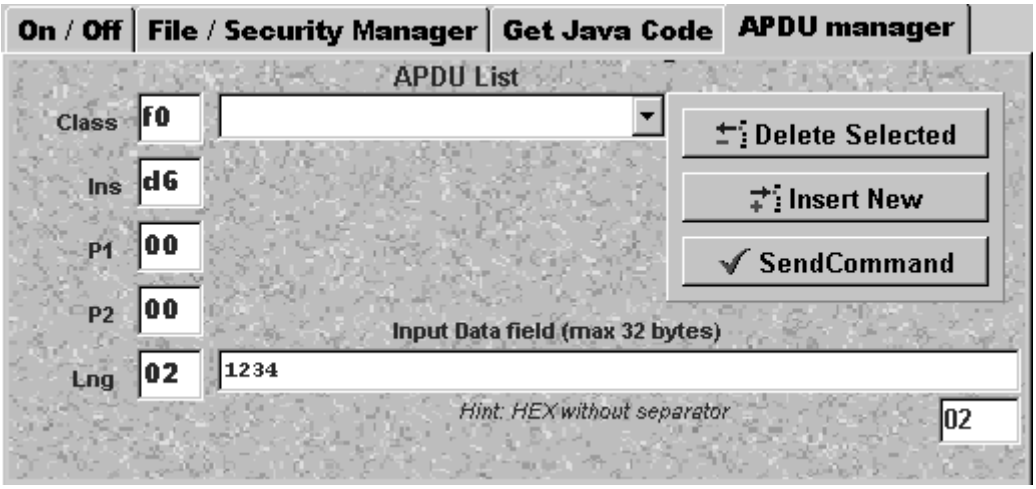
2. A második mező a file-műveleteket tartalmazza. A baloldali rész file létrehozására, a jobb felső rész a titkos kulcsok kezelésére szolgál. A jobb alsó részben található a file kiválasztás (select), törlés (delete), illetve a könyvtárak megjelenítésére szolgáló funkciók.



3. A harmadik panel egy program letöltésére, és elindítására szolgál. Itt miután betöltöttük a bináris programkódot a memóriába (középső szövegmező) rátölthetjük a kártyára (előzőleg a letöltés cél-file-ját a file-kezelő panel segítségével ki kell jelölni), majd beállíthatjuk mint a következő indítás utáni aktuális program.



4. A negyedik panel segítségével küldhetünk olyan parancsokat, amelyek nem szerepelnek az előző menükben, vagy esetleg nem is az alapértelmezett betöltőnek szólnak (a Cardlet definiálhat saját parancsokat). Itt külön mezőben beállítható a parancs öt byte-ja, és az alsó hosszabb mezőben megadható a parancs adatrészének tartalma is.



A specifikáció

A kártyás rendszerek egyik sarkalatos pontja az azonosítás. A kártyát tulajdonképpen minden rendszer arra használja fel, hogy a felhasználót azonosítsa, a felhasználóra vonatkozó valamilyen adatokat tárolja, majd ezen információk segítségével különféle szolgáltatásokat nyújtson. A szolgáltatásokat csak a valódi felhasználó veheti igénybe, ezért az adatokat megfelelő módon védeni kell az illetéktelen felhasználókkal szemben. Egy alkalmazás implementációjakor figyelembe kell venni a kártya korlátos erőforrásait, és a szolgáltatás természetétől függően meg kell osztani a terheket a kártya illetve egy központi szervergépen fenntartott adatbázis között. A fejlesztőnek az alkalmazások elkészítésekor döntenie kell:

- A kártyán tárol minden adatot, mert azok mérete ezt lehetővé teszi. A szolgáltatás igénybevételekor a felhasználó csak a terminállal kommunikál, nincs szükség semmilyen távoli kapcsolatfelvételre. Az adatok helyben maradnak, a művelet biztonságos, és gyors.
- Minden adatot egy szervergép adatbázisában tárol, mert ezek olyan nagyok, hogy nem férnének el a kártyán, vagy esetleg olyan adatokról van szó, amely a kártyát felhasználó személy és egy nagyobb szolgáltató közötti kapcsolatot írnak le. (Például egy banki szolgáltatás esetén az ügyfél bankszámlájára vonatkozó adatok.)
- Valamilyen köztes megoldást választ. Megosztja az adatokat. Ekkor egyszerűbb műveleteket a terminál is el tud végezni, és esetleg bonyolultabb műveletekhez szükséges valamilyen távoli kapcsolatfelvétel. (Ilyen megoldást alkalmaznak a különböző elektronikus pénztárcák esetén: kisebb összegek esetén a kártyán található összegből fizet az ügyfél; nagyobb összeg, vagy a kártyán található készpénz kifogyása esetén a terminál felveszi a kapcsolatot a bank adatbázisával, és arról tölti fel a kártyát.)

Egy kártyán a megfelelő szabványok betartása mellett (néhány területen ezek a szabványok még a kidolgozásra várnak) és a szolgáltató cégek egyessége alapján több szolgáltatás is lehet. Semmilyen elvi akadálya nincs annak, hogy egy kártya több egymástól független funkcióval is rendelkezzen.

A szakdolgozat keretein belül egy demonstrációs program elkészítését tűztük ki célul. Ezzel kívánjuk szemléltetni a kártyában rejlő lehetőségeket. Egy egyszerű felhasználás implementálásán keresztül szeretnénk bemutatni egy kártyás rendszer működési mechanizmusát. A rendszer tervezésekor, a második modellt választottuk. A kártya tulajdonképpen csak a felhasználó (ez esetben a diák) azonosítására szolgál. A kártya csak a hallható nevét és egy azonosítót tárol. Ez alapján tartja őt számon a központi adatbázis, így a tulajdonképpeni szolgáltatást a szervergép nyújtja. Az azonosítás az PIN (Personal Identification Number), azaz, ha az aki használja a kártyát ismeri ezt a számot, akkor valóban jogosult a szolgáltatást igénybe venni. A PIN kódot a kártyán tároljuk.

Az én feladatom volt a terminál és a kártya közötti kommunikáció megvalósítása. Amikor a hallgató odalép a terminálhoz, és bedugja a kártyát az olvasóba, a terminál megkéri, hogy írja be a PIN kódját. Ez után üzenetet küld a kártyának, hogy szüksége lenne a hallgatót a szerveren egyértelműen azonosító sorszámra. Az üzenetben szerepelnie kell a PIN kódnak. Amennyiben a

PIN kód megegyezik a kártyán tárolt PIN kóddal, a kártya visszaküldi a kért adatot, amennyiben nem jó a kód, ezt jelzi a terminálnak.

Az elkészült java osztály metódusait a társam által készített hálózati alkalmazás használja majd.

A funkciók

1. Kell lennie egy metódusnak, amely végrehajtja a következő metódusokat:
    - Elindítja a kártyát.
    - Közli a kártyával, hogy melyik alkalmazást kívánjuk igénybe venni.
    - Elküldi a kérést és a PIN kódot.
    - Elemzi a választ, nyomtatható formára alakítja az adatokat, vagy megfelelő formában jelzi az esetleges hibákat.
    - Az olvasót újra kikapcsolja.

2. A program lehetőséget kell nyújtson a PIN kód, a felhasználói név illetve az azonosító be- és átállítására. Ennek a mechanizmusa megegyezik az előző funkciónál leírtakkal. Ehhez a művelethez is az érvényes PIN kód ismerete lesz majd szükséges.
3. A programnak minden kártyával kapcsolatos hibát is jeleznie kell. Ezt a Java-s felhasználás miatt célszerű kivételekkel (Exception) megoldani. A kivétel lekezelése folyamán megállapítható legyen, hogy mi okozhatta a hibát, és hogy a hibának milyen következményei vannak (az olvasót sikerült-e kikapcsolni, stb.).

Az adatok

1. A PIN kód négyjegyű szám
2. A név legfeljebb 32 karakterből álló karaktersorozat
3. Az azonosító 14 byte-os

Határfelület

A program metódusait egy Java applikáció fogja felhasználni, ezért célszerű valamilyen java osztály formájában implementálni

A programnak lesz egy olyan eljárása is, amelynek segítségével a funkciók parancs üzemmódból (prompt-ból) is elérhetőek. Ennek lehetőleg legyen olyan funkciója, amely segítségével elemi üzeneteket illetve üzenetek sorozatát lehet küldeni a kártyának.

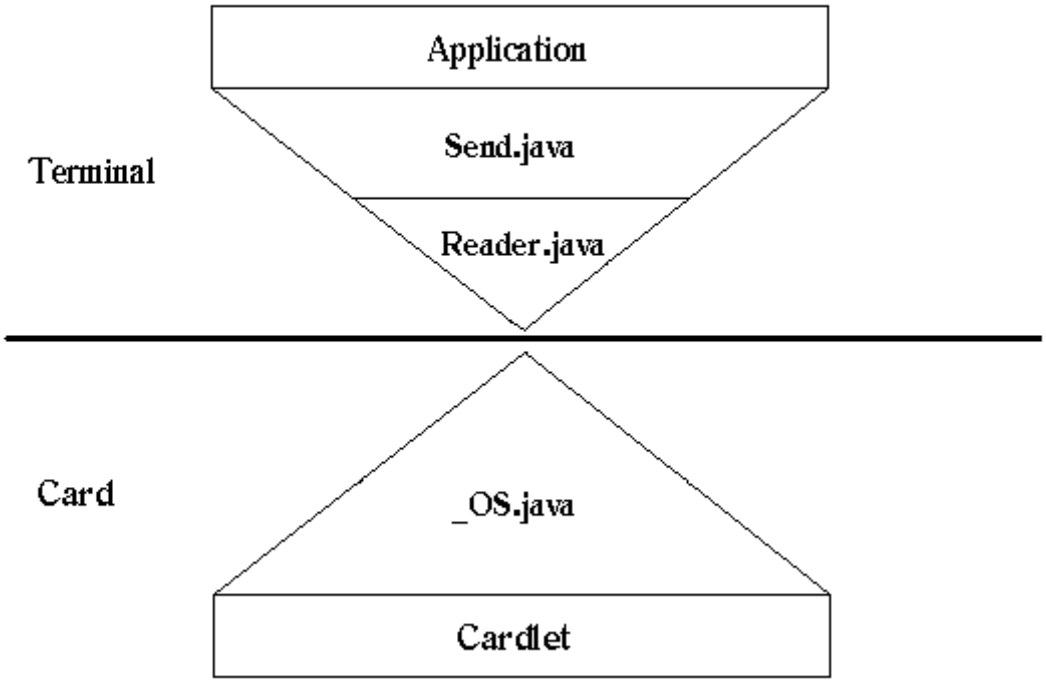
Korlátozások

A program hiba esetén nem tudja azt elhárítani, csak jelezni fogja a hibát kiváltó okot. Például egy írás parancs végrehajtása közben érkezet hiba esetén nem lehet a kivétel alapján megállapítani, hogy mennyi adatot sikerült ráírni a kártyára, és mi az ami elveszett.

A program kénytelen a „port”-ot használni, ebből kifolyólag nem lehet majd Java „Applet”-ből, csak „Applikáció”-ból használni.

Az elkészült program leírása

A kommunikációt a következő ábra szemlélteti:



Az elkészült program két részből áll:

- A kártyán található, és a kártya virtuális gépe által futtatható Cardlet, amely megoldja a szükséges file és ellenőrző műveleteket.
- A terminálon futó program, amely a kártyával történő kapcsolatot valósítja meg. Ezt használja majd a társam által elkészített hálózati alkalmazás, ami ugyancsak a terminálon illetve egy szervergépen fog futni.

A Cardlet leírása

A program Java kódját tartalmazó állomány neve:	pin.java
A java osztály neve:	pin
A bináris file mérete:	518 byte
A helye a kártyán (file azonosító)	3333
A Cardlet adatait tartalmazó file	3331

A Cardlet az ISO 7813 szabványnak megfelelően fogadhatja az üzeneteket.

Osztály	Utasítás	1. Paraméter	2. Paraméter	Adat hossza
CLA	INS	P1	P2	LNG

Az üzenet parancs része öt byte-os. Ennek első és második byte-ja határozza meg az utasítást. Erre a célra bevezettem egy saját utasítás osztályt illetve a két műveletnek megfelelő utasításokat jelző byte-okat:

- A név és az azonosító lekérdezése esetén:

CLA=33 INS=55

- A név, az azonosító és a PIN kód beállításakor:

CLA=33 INS=66

Ezek természetesen hexadecimális értékek.

A két átadható paraméter minden esetben a művelet végrehajtásához szükséges PIN kódot tartalmazza:

P1= első két jegy P2= második két jegy

A PIN kódot mindig két byte-on ábrázoltam, ahol az első byte hexadecimális képe felel meg a PIN kód első két számjegyének, és a második byte a második két számjegynek.

Az adatok hossza a lekérdezés esetén nulla, mert nincs egyéb paraméter; a név és az azonosító beállításakor pedig 30 (hexadecimálisan: 16*3=48).

A Cardlet-hez tartozó file-ok a következők:

- A Cardlet kódját a 3333 azonosítóval rendelkező file tartalmazza, amelynek hosszát az mksolo program segítségével határoztuk meg, a bináris állomány elkészítésekor (mksolo –s pin.class). A file-ra senkinek nincs olvasás joga, és miután felkerült rá a program meg kell szüntetni a rá vonatkozó írásjogot is. Ezt a második lépést a kártyák újrafelhasználhatósága érdekében nem tettem meg. A teljes biztonság érdekében ezeket a file-okat egy alkönyvtárba kéne tenni, hogy megtilthassuk a törlést. (A törléshez egyébként mindig mindenkinek van joga, ha van törlésjoga az aktuális könyvtárhoz. A programot tartalmazó file letörlése majd újra létrehozása esetén megint ott lennénk „ahol a part szakad”, hisz akárki átírhatná a Cardlet-et, a file méretkorlátain belül, és így hozzáférhetne a rejtett adatokhoz)
- A Cardlet adatait tartalmazó file, amelynek tartalmát a következő táblázat szemlélteti:

0		0A
A file fejléce (16 byte)		
0		0A
PIN (2 byte)	ID (14 byte)	
10		1A
Név (2*16 byte)		
20		2A
Név		

Ez a file csak „supervisor” joggal hozzáférhető. A file a fejléccel együtt 64 byte hosszú.

Ezek után a funkciók végrehajtása a következő műveleteket jelenti:

- A név és az azonosító lekérdezésekor a parancs paramétereit össze kell hasonlítani a 3331 file első két byte-jával, majd egyezés esetén el kell küldeni a file tartalmát az olvasónak.
- A név, azonosító ill. PIN kód beállításakor a régi PIN kód egyezése esetén át kell írni a 3331 file tartalmát.

A terminálon futó program

A program Java kódját tartalmazó file neve:	Send .java
Az osztály neve:	Send
Az olvasás műveletet végrehajtó eljárás neve:	GetNameAndID
Az írást végrehajtó eljárás neve:	SetNameAndID
A parancs üzemmódot megvalósító eljárás:	main és Execute

A Send osztály

```
public static void GetNameAndID(
                String PIN,          a PIN kódot tartalmazó karakterlánc
                StringBuffer name,   a név értéke ebbe kerül majd
                StringBuffer id)     az azonosító értéke ebbe kerül majd
                throws STUDException hiba esetén kivétel váltódik ki

public static void SetNameAndID(
                String oldPIN,       régi PIN kód
                String newPIN,       új PIN kód
                StringBuffer name,   az új név
                StringBuffer id)     az új azonosító
                throws STUDException hiba esetén kivétel váltódik ki
```

A parancssorból végrehajtható utasítások:

- GET Paraméterként a PIN kódot kell megadni. Ha nincs PIN kód, az alapértelmezett „1234” kódot használja.
- SET Kötelezően négy paramétere van: régi PIN kód, új PIN kód, új azonosító, új név. Az azonosító kötelezően 28 jegyű hexadecimális kell legyen, a névnek pedig az első 32 karaktere kerül a kártyára. Ennél az alkalmazásnál sajnos nem lehet szóköz karakter, mert akkor a nevet két külön paraméternek tekintené az operációs rendszer.
- Lehet ezenkívül hexadecimális számsorozatokkal megadott byte-okat is küldeni a kártyának. A számsorozatok között a „RESET” parancs is használható.

Kivételek

A kártyával kapcsolatos hibákat a STUDException kivétel segítségével kezelem.
Mezői:

- Id — A kártya által adott két byte-os hibakódot tartalmazza. Az „FFFF” érték azt jelzi, hogy nem a kártya generálta a hibát, hanem a kommunikáció szakadt meg valahogyan. Ekkor a „comment” mezőből lehet eldönteni, hogy a folyamat mely pontján szakadt meg a kapcsolat.
- Comment — Ez a mező tartalmazza szöveges formában a hiba okát.
- ReaderOff — Ez egy logikai értéket tartalmaz, amely igaz, ha a hiba ellenére sikerült az olvasót kikapcsolni, ill. hamis, ha az olvasó nincs kikapcsolva (ekkor ezt más úton kell eszközölni).

```
/**
* File: Send.java
* Date: 98.06
*/

public class Send implements Constants{

      final static String MYGETCOMM=new String("3355");
      final static String MYSETCOMM=new String("3366");
      final static String MYAPPFILE=new String("3333");

      final static int WRONG_PIN=0x6b10;

      static Reader reader;

      /**
      *This is the procedure, to determine the ID and the holder's name
      *@param PIN the pin code
      *@param name to this StringBuffer appends the name of card holder
      *@param id to this StringBuffer appends the card id
      *@exeption in case of any error, allso in case of wrong pin code
      */
      public static void GetNameAndID(
                  String PIN,
                  StringBuffer name,
                  StringBuffer id)
                  throws STUDException{

            if(PIN.length()!=4)
                  throw new STUDException("Wrong pin code length");

            //variables:
            reader=new Reader();
            StringBuffer response=new StringBuffer("");
            StatusWord statusword=new StatusWord();

            //init the reader:
            try{
                  reader.ReaderAllocate("Reflex60","COM2");
                  reader.ReaderPowerUp();
                  reader.ReaderReset(response);
            }catch(Exception e){
                  showerror("ffff","Can't allocate the reader\n");
            }

            //init my cardlet:
            try{
                  response=new StringBuffer("");
                  //System.out.println("Send:F0F2"+MYAPPFILE+"010F");
                  reader.ReaderSendIsoOutT0("F0F2"+MYAPPFILE+"010F",
                        response,statusword);
            }catch(Exception ex){
                  showerror("ffff","Can't init the cardlet\n");
            }

            //test the ST.SUCCESS
            if(!response.toString().equals("9000"))
```

```
                showerror(response.toString(),"Error at init cardlet:");

          //RESET:
          try{
                response=new StringBuffer("");
                //System.out.println("RESET");
                reader.ReaderReset(response);
          }catch(Exception ex){
                showerror("ffff","Can't RESET the card\n");
          }

          //communication with my cardlet:
          try{
                response=new StringBuffer("");

                //System.out.println("Send:"+
                      //(MYGETCOMM+PIN+"00").toUpperCase());
                       reader.ReaderSendIsoOutT0(
                      (MYGETCOMM+PIN+"00").toUpperCase(),
                            response,statusword);
          }catch(Exception e){
                showerror("ffff","Problem with cardlet\n");
          }

          //test the ST.SUCCESS
          String sw=response.toString();
                //cat the last 4 digits:
          sw=sw.substring((sw.length()<4)? 0 : sw.length()-4 ,sw.length());
          if(!sw.equals("9000"))
                showerror(sw,"Error at send command:");

          //transform the result and append
          //  to id and name StringBuffers:
          id.append(response.toString().substring(4,2*0x10));
          String str=response.toString().substring(2*0x10,2*0x30);
          for(int i=0;i<2*0x20;i+=2)

      name.append((char)Integer.parseInt(str.substring(i,i+2),16));

          //turn off the reader
          try{
                reader.ReaderFree();
          }catch(Exception e){
                throw new STUDException("Can't deallocate the
reader\n",false);
          }
      }


      /**
      *This is the procedure, to set the pin code, Id and the holder's name
      *@param oldPIN the actual pin code
      *@param newPIN the new pin code
      *@param the new name
      *@param the new id
      *@exception in case of error or wrong pin code
      */
```

```
public static void SetNameAndID(
            String oldPIN,
            String newPIN,
            StringBuffer name,
            StringBuffer id)
            throws STUDException{

      if(oldPIN.length()!=4)    //pin's
            throw new STUDException("Wrong old pin code length");
      if(newPIN.length()!=4)
            throw new STUDException("Wrong new pin code length");

      if(id.length()!=2*14)     //id
            throw new STUDException("Wrong Id length");

            //update newdata:
      StringBuffer newdata=new StringBuffer(newPIN);
      newdata.append(id.toString().toUpperCase());
      for(int i=0;i<0x20;i++){    //name
            if(name.length()>i){
                  newdata.append(
                        Integer.toHexString(
                        (int)name.charAt(i) ));
            }else{
                  newdata.append("00");
            }
      }
      //System.out.println("New data:"+newdata);

      //variables:
      reader=new Reader();
      StringBuffer response=new StringBuffer("");
      StatusWord statusword=new StatusWord();

      //init the reader:
      try{
            reader.ReaderAllocate("Reflex60","COM2");
            reader.ReaderPowerUp();
            reader.ReaderReset(response);
      }catch(Exception e){
            showerror("ffff","Can't allocate the reader\n");
      }

      //Init my cardlet:
      try{
            response=new StringBuffer("");
            //System.out.println("Send:F0F2"+MYAPPFILE+"010F");
            reader.ReaderSendIsoOutT0("F0F2"+MYAPPFILE+"010F",
                  response,statusword);
      }catch(Exception ex){
            showerror("ffff","Can't init the cardlet\n");
      }

      //test the ST.SUCCESS
      if(!response.toString().equals("9000"))
            showerror(response.toString(),"Error at init cardlet:");
```

```
            //RESET:
            try{
                  response=new StringBuffer("");
                  //System.out.println("RESET");
                  reader.ReaderReset(response);
            }catch(Exception ex){
                  showerror("ffff","Can't RESET the card\n");
            }

            //communication with my cardlet:
            try{
                  response=new StringBuffer("");
                  //System.out.println("Send:"+
                        //(MYSETCOMM+oldPIN+"30"+newdata).toUpperCase());

                  reader.ReaderSendIsoInT0(
                        (MYSETCOMM+oldPIN+"30"+newdata).toUpperCase(),
                        response,statusword);
            }catch(Exception e){
                  showerror("ffff","Problem with cardlet\n");
            }

            //test the ST.SUCCESS
            String sw=response.toString();
            sw=sw.substring((sw.length()<4)? 0 : sw.length()-4 ,sw.length());
            if(!sw.equals("9000"))
                  showerror(sw,"Error at send command:");

            //turn off the reader
            try{
                  reader.ReaderFree();
            }catch(Exception e){
                  throw new STUDException("Can't deallocate the
reader\n",false);
            }
      }

      /**
      *Throws an exception in case of error
      *@param sw the status word returnd from card
      *@param prefix the prefix of exceptions comment, this will be the hole
comment in case of another exception
      */
      public  static void showerror(String sw,String prefix) throws
STUDException{

            STUDException exi;
            switch(Integer.parseInt(sw,16)){
            case 0x6a82:
                  exi= new STUDException(prefix+
                  "File not found",0x6a82);
                  break;
            case 0x6b00:
                  exi= new STUDException(prefix+
                  "Incorrect parameters",0x6b00);
                  break;
            case 0x6a80:
```

```
                exi= new STUDException(prefix+
                "Invalid file type",0x6a80);
                break;
          case 0x6300:
                exi= new STUDException(prefix+
                "No action performed",0x6300);
                break;
          case 0x6982:
                exi= new STUDException(prefix+
                "No privilage to execute the application",0x6982);
                break;
          case WRONG_PIN:
                exi= new STUDException(prefix+
                "Wrong PIN code",WRONG_PIN);
                break;
          case 0xffff:
                exi= new STUDException(prefix);
                break;
          default:
                exi= new STUDException(prefix+
                sw,Integer.parseInt(sw,16));
          }

          //turn off the reader:
          try{
                reader.ReaderFree();
                exi.ReaderOff=true;
                throw exi;

          //in case of error set the ReaderFree field false:
          }catch(SLBAPIAbortException ex){
                exi.ReaderOff=false;
                throw exi;
          }catch(SLBAPIErrorException ex){
                exi.ReaderOff=false;
                throw exi;
          }catch(SLBAPITimeOutException ex){
                exi.ReaderOff=false;
                throw exi;

          }

    }


//---------------------------------------------------------------------


    /**
    * The main method for command prompt use, see the usage message.
    */
    public static void main(String[] args){
          String PIN;

          //if no parameter
          if(args.length<1){
                System.err.println(
```

```
                              "Usage: java Send <the command APDUs>\n"+
                              "       java Send GET [<pin code>]\n"+
                              "       java Send SET <old pin code> <new pin
code> <id> <name>\n");
                  System.exit(1);

            //execute GET:
            }else if(args[0].equals("GET")){
                  if(args.length>2) {
                        System.out.println("Invalid number of arguments");
                        System.exit(1);
                  }

                  if(args.length==1) {  //if no pin code
                        System.out.println("Using default PIN:1234");
                        PIN="1234";
                  } else PIN=args[1];

                  StringBuffer name=new StringBuffer(""),
                        id=new StringBuffer("");
                  try{
                        GetNameAndID(PIN,name,id); //get name and id
                        System.out.println(
                              "\nName: "+name+
                              "\nId:   "+id);
                  }catch(STUDException e){
                        System.out.println(e.toString()+", id:"+
                              Integer.toHexString(e.id));
                  }

            //execute SET:
            }else if(args[0].equals("SET")){
                  if(args.length!=5){
                        System.out.println("Wrong number of argument");
                        System.exit(1);
                  }

                  try{
                        SetNameAndID(
                              args[1],//old pin
                              args[2], //new pin
                              new StringBuffer(args[4]),//name
                              new StringBuffer(args[3]));//id
                        System.out.println("Name and id changed");
                  }catch(STUDException e){
                        System.out.println(e.toString()+", id:"+
                              Integer.toHexString(e.id));
                  }

            //send the APDUs
            }else {
                  StringBuffer result=new StringBuffer();
                  System.out.println(result=Execute(args));
            }
      }

      /**
```

```
	* To help the main method.
	*@param args the String array with the commands
	*@return the messages: {<command>--><respons>} [<error>]
	*/
	public static StringBuffer Execute(String[] args){
		String command=args[0];
		Reader reader=new Reader();
		StringBuffer ret=new StringBuffer("");

		//init the reader:
		StringBuffer response=new StringBuffer("");
		StatusWord sw=new StatusWord();
		try{
			reader.ReaderAllocate("Reflex60","COM2");
			reader.ReaderPowerUp();
			reader.ReaderReset(response);
		}catch(Exception e){
			ret.append("Can't allocate the reader\n");
			return ret;
		}

		//send the commands:
		for(int i=0;i<args.length;i++)
		try{
			command=args[i];
			response=new StringBuffer("");

			if(command.equals("RESET")){
				reader.ReaderReset(response);
			}else{
				reader.ReaderSendIsoOutT0(command.toUpperCase(),
					response,sw);
				}
			ret.append(command+"--
>"+response/*+":"+sw.toString()*/+"\n");
		}catch(Exception e){
			ret.append("Can't send the command\n");
			return ret;
		}

		//turn off the reader
		try{
			reader.ReaderFree();
		}catch(Exception e){
			System.out.println("Can't deallocate the reader\n");
		}
		return ret;
	}
}
```

```
/**
* File: StudException.java
* Date: 98.06
*/
import java.lang.Exception;

public class STUDException extends Exception{
      public int id=0xffff;
      String comment="Unknown error";
      public boolean ReaderOff=true;

      public STUDException()
      {
            super();
      }

      public STUDException(String comm)
      {
            super(comm);
            comment=comm;
      }

      public STUDException(String comm, int i){
            super(comm);
            id=i;
            comment=comm;
      }

      public STUDException(String comm, boolean state){
            super(comm);
            comment=comm;
            ReaderOff=state;
      }
}
```

```
/**
* File: pin.java
* Date: 98.06
*/

import javacard.framework.*;
import javacardx.framework.*;

public class  pin implements ST, ISO {

  static final byte ACK_SIZE                = (byte)1;
  static final byte ACK_CODE                = (byte)0;

  static final byte MYAPPCLASS               =(byte)0x33;
  static final byte GETINS             =(byte)0x55;
  static final byte SETINS                  =(byte)0x66;
  static final byte SETPININS             =(byte)0x77;
  static final byte WRONG_PIN               =(byte)0xAA; //6B10


  public static void main (String args[]) {

    byte pbuffer[]=new byte[ISO.COMMAND_LENGTH];
    byte dbuffer[]=new byte[0x30];
    byte ackByte[]=new byte[ACK_SIZE];

    _OS.Execute((short)0, (byte)0);

    byte sw=(byte)0;

    _OS.GrantSupervisorMode();

    do {

      _OS.GetMessage(pbuffer, ISO.COMMAND_LENGTH, ACK_CODE);

      if (pbuffer[0] != MYAPPCLASS){
          sw= ST.INVALID_CLASS;
      }else if ((pbuffer[1]!=GETINS) && (pbuffer[1]!=SETINS) &&
(pbuffer[1]!=SETPININS)){
        sw= ST.INS_NOT_SUPPORTED;
      }else if ((pbuffer[4]!=0) && (pbuffer[1]==GETINS)) {
          sw= ST.INVALID_PARAMETER;
      }else if ((pbuffer[4]!=0x30) && (pbuffer[1]==SETINS)){
        sw= ST.INVALID_PARAMETER;
      }else {

        _OS.SelectFile((short)0x3331);
        _OS.ReadBinaryFile((short)0x0000, (byte)0x30, dbuffer);

        if((pbuffer[2]==dbuffer[0]) && (pbuffer[3]==dbuffer[1])){
             if(pbuffer[1]==GETINS){

            ackByte[0]=pbuffer[1];
            _OS.SendMessage(ackByte, ACK_SIZE);
            _OS.SendMessage(dbuffer,(byte)0x30);
```

```
                sw=ST.SUCCESS;
            }else {
             _OS.GetMessage(dbuffer, (byte)0x30, pbuffer[1]);

             //_OS.SelectFile((short)0x3331);
             _OS.WriteBinaryFile((short)0x0000, (byte)0x30, dbuffer);
             sw=ST.SUCCESS;
            }
          } else {
             sw=WRONG_PIN;
          }
        }

        _OS.SendStatus(sw);
      } while (true);
    }
}
```

Irodalomjegyzék

- Cyberflex Kit programozói kézikönyv, Schlumberger
- Intelligens Kártya Hírlevél 1997. Október
- Implementing the Student Chip Card in Holland’s Universities, CardTech/SecureTech ’97, Orlando, Florida, Piet Maclaine Pont