行政院所屬各機關因公出國人員出國報告書

(出國類別：考察)

至德國 G&D 公司進行健保 IC 卡 Applet 程式撰寫及監督測試之技術移轉

心得報告

服務機關：中央健康保險局

出國人　職　稱：科員、課員

　　姓　名：孫浩淳、張榮宗

出國地點：德國

出國期間：91 年 1 月 8 日至 91 年 2 月 5 日

報告日期：91 年 5 月

系統識別號:C09100509

公 務 出 國 報 告 提 要

頁數: 67　含附件: 否

報告名稱:

前往德國G&D公司進行健保IC卡Applet程式撰寫及監督測試之技術移轉

主辦機關:

行政院衛生署中央健康保險局

聯絡人／電話:

劉彥秀／27029959

出國人員:

孫浩淳　行政院衛生署中央健康保險局　資訊處　科員
張榮宗　行政院衛生署中央健康保險局　中區分局　課員

出國類別: 考察

出國地區: 德國

出國期間: 民國 91 年 01 月 08 日 -民國 91 年 02 月 05 日

報告日期: 民國 91 年 05 月 10 日

分類號/目: J0／綜合（醫藥類）　I8／資訊科學

關鍵詞:　健保IC卡,Java,Java Card Applet

內容摘要: 我國即將在九十一年七月起，陸續換發新一代的中華民國健保IC卡，整合現行的各類健保就醫憑證，取代行之有年的紙卡。 在健保IC卡計畫中，未來所使用的卡片是目前相當先進的JAVA CARD智慧卡，在得標廠商東元公司的安排下，委由德國G&D SECARTIS公司設計JAVA CARD APPLET程式，中央健保局則派員前往當地觀摩學習並作技術移轉。本次出國期間不但學習到一些程式設計及測試工具軟體的功能，並參與了部分程式開發設計及測試作業，及和該公司相關人員充分討論及交換意見。 相關心得及報告則可作爲持續完成本計畫及未來自行改良的參考。

本文電子檔已上傳至出國報告資訊網

行政院所屬各機關因公出國人員出國報告書

(出國類別：考察)

至德國 G&D 公司進行健保 IC 卡 Applet 程式撰寫及監督測試之技術移轉

心得報告

服務機關：中央健康保險局

出國人　職　稱：科員、課員

　　姓　名：孫 浩 淳、張 榮 宗

出國地點：德國

出國期間：91 年 1 月 8 日至 91 年 2 月 5 日

報告日期：91 年 5 月

摘要

我國即將在九十一年七月起，陸續換發新一代的中華民國健保 IC 卡，整合現行的各類健保就醫憑證，取代行之有年的紙卡。

在健保 IC 卡計畫中，未來所使用的卡片是目前相當先進的 JAVA CARD 智慧卡，在得標廠商東元公司的安排下，委由德國 G&D SECARTIS 公司設計 JAVA CARD APPLET 程式，中央健保局則派員前往當地觀摩學習並作技術移轉。本次出國期間不但學習到一些程式設計及測試工具軟體的功能，並參與了部分程式開發設計及測試作業，及和該公司相關人員充分討論及交換意見。

相關心得及報告則可作為持續完成本計畫及未來自行改良的參考。

目次

壹、目的

由於資訊時代的來臨，為了提昇全民健康保險的服務和品質，我國即將在九十一年七月起，陸續換發新一代的中華民國健保 IC 卡，整合現有的多種健保就醫憑證，以取代行之有年的紙卡，來提供民眾更方便、更安全的保險憑證。

因為這項健保 IC 卡計畫的成敗攸關社會大眾福祉和民眾就醫權益，所以必須規劃一套完善而可行的資訊系統，才能處理將來所衍生的龐大資料。本局經由得標廠商東元公司的協助，委請世界知名的智慧卡大廠-德國 G&D 公司負責設計健保 IC 卡上的 Javacard applet 程式，因為這項先進技術在國內的應用實例尚屬極少數，本局乃藉此機會協調派員前往學習，並作必要的技術移轉及溝通討論，以利未來自行開發及改良。

貳、 參觀學習過程紀要

一、*Java 和 Javacard*

在國民健保 IC 卡計畫中，未來所採用的卡片種類就是我們耳熟能詳的智慧卡(smartcard)，但是在實體設計上則是其中最新的 Javacard；卡片上實際運作的程式是所謂的 Java card applet，其規格和定義則屬於 J2ME (Java 2 micro edition)的子集(subset)，也就是從 Java 標準規格中萃取一些針對智慧卡所特別設計的功能和函式庫（可以統稱為 API），當然它還是符合了 Java 的基本架構。因此本報告的主題雖然並不著重在 Java 語言的介紹（但是後面章節會介紹 Java card applet 語言），但是還是要簡單了解一下 Java 的昨日，今日和明日。

Java 是一個由美國昇陽(Sun)公司在 90 年代所發展出來的新一代程式語言，嚴格來說，從 Java 1.0 定案到今天，它的歷史還不到 8 年，只是個二年級的小學生而已，但是到目前為止，它對資訊產業所帶來的影響卻不亞於任何前輩如 c、BASIC..等久富盛名的程式語言，它的茁壯，當然受到網際網路這幾年來的大量普及和應用的推波助瀾有莫大關係；或者換個角度來看，它的誕生也正是針對網路虛擬世界的特殊需求而來。

因為網路的最大特色就是結合了不同種類，不同作業系統的電腦，在不同的時空下達到無障礙的溝通環境。因此 Java 語言在設計的時候，一個重大的目標就是要作到在一個有著不同機器，不同作業系統的分散式網路環境下，都能夠以最小的成本開發軟體，也就是所

謂的"跨平台"，換句話說，一個以 Java 語言開發出來的應用程式，即使機器平台從執行 Windows 視窗作業系統的個人電腦改換到執行 UNIX 的伺服器主機上，也不需要作任何修改即可正常執行。當然，前提是在這二種機器上都存在 JVM(Java Virtual Machine)的虛擬執行環境。

正因為有了 Java 和 JVM，使得跨平台的理念得以落實，可攜性也大幅提高，從此一套程式碼可以行遍天下，也不需要再為如"這須視你所使用的編譯程式而定"，或是"關於這個函數，不同的機器上可能有不同的定義"等不確定因素，而在不同的平台上修改程式，以確保它能正常運作.

除此之外，JVM 還有一個好處，因為 Java 必須透過 JVM 才能運作，JVM 便可以有效扮演把關者的角色，對要執行的 Java 程式碼作必要的安全檢查，保證程式內容並沒有違反任何安全控管權限，使用不當的呼叫程序，或是參考或修改任何不屬於它應用的範圍， 因為 Java 語言本身雖然在設計上已經拿掉了一些參照記憶體空間的功能，如指標(pointer)，但是並不能排除有心人撰寫像病毒類的惡意的程式在網路上散播，而 JVM 剛好可以作為最後一道有效的防火牆。

當然 Java 也不是沒有缺點，譬如說因為它多了一些檢查的機制，又必須透過 JVM 的機制下才能執行，難免會犧牲一些效率和增加程式開發的難度，但整體而言，在網路安全為首要考量的前提下，這些代價是值得的。雖然有人認為，網路無法真正作到百分之百的安全，但是在比較各種作法，以及經過不斷的檢驗和討論之後，Java 相對而言仍然是一個比較安全的語言。

隨著 Java 應用的普遍度及接受度越來越高，Sun 公司也努力的將 Java 推向各種不同的資訊平台，例如 IA 產品，PDA，行動電話、智慧卡....等，再由相關硬軟體廠商針對不同的產業如金融、醫療、通訊、大眾運輸..等設計不同的商品，包括這次我國即將換發的健保 IC 卡，也是 Java 應用下的產物。

有關 Java，JVM，Java card applet 的關係，會在後面的"開發 Java card applet 程式"章節有進一步的說明。

二、*Java Card Applet 程式開發*

(一)Sm@rtcafe 軟體簡介

Sm@rtcafe 此套整合式視窗軟體為德國 G&D 公司專門用來設計 G & D Java card Apllet (G & D Java card 簡稱 GDJC)的開發工具軟體 (Java Card Applet development Software)。

它能模擬包括下列各項真的 GDJC 的完整行為

- Bytecode interpreter
- Java Card Runtime Environment (JCRE)
- Installed APIs (Application Programming Interfaces)
- Java Card hardware

Sm@rtcafe 提供給程式開發者，可用來測試及偵錯 applets， 其 applet 模擬執行的環境幾乎相同於卡片上真正的運作環境，它並提供完整透明的 applets 執行時，每一命令動作時所發生的所有行為記錄; 它也讓你在測試執行各種不同的 applet 時，提供相關資源配置情形及執行效能上所表現的數據。

Sm@rtCafe 的功能性包含：

- 載入套件(package)到模擬的 EEPROM 記憶體中(包括位址定位)和對應到使用的記憶體
- 以架構圖顯示套件的資料成員和方法
- 可以同時顯示 bytecode 和原始碼
- 支援在除錯模式下，用步階方式執行 JavaCard 的程式碼，並可設定程式碼或函式暫停點(breakpoint)
- 可以顯示正在執行中 JavaCard 的程式碼的變數內容或狀態
- 可以使用和呼叫 stack

- 可使用區域變數
- 可以顯示在 heap 上的 instance 配置情形
- 可以顯示目前資源消耗情形(resource consumption)，也就是 stack 和 heap 記憶體使用情形
- 追蹤(Tracking) 例外事件處理函式（exception events）執行情形
- 轉換及編輯 APDU 命令集
- 可從 script 檔中定義參數化的 APDU 命令
- 轉換 applet class 格式檔成為 CAP 格式檔
- 可任意連接 G&D 端末設備來下載 Java Card 上的類別物件 (implicit CAP conversion)
- 可以在實際的硬體上執行 command script
- 記錄所有執行命令及回應結果於記錄簿中(logbook)

(二)Sm@rtcafe 軟體安裝

安裝 Sm@rtcafe 前，需先安裝 java sdk(至http://www.sun.com/java 下載) 及 javaca rd api 環境 (至http://www.sun.com/javacard 下載)

1. 執行 j2sdk-1_3_1_01-win.exe(java sdk)
   (例如安裝至 c:\jdk1.3.1_01\)
2. 解壓縮 java_card_kit-2_1_2-win.zip(javacard api)
   (例如安裝至 c:\java_card_kit-2_1_2\)
3. 利用光碟片，安裝 Sm@rtcafe (steup.exe)
4. 設定 java applet complier 環境的 bat 檔(可利用記事本編輯新檔，檔名為 build.bat)， 其目的用來產生 javacrd applet class

build.bat 內容如下(需依據實際 Jdk 及 Javapcard Api 所安裝目錄來設定)

```
set _CLASSES=C:\java_card_kit-2_1_2\lib\apduio.jar;
 C:\java_card_kit-2_1_2\lib\apdutool.jar;C:\java_card_kit-
 2_1_2\lib\jcwde.jar;C:\java_card_kit-2_1_2\lib\converter.jar;C:\java_card_kit-
 2_1_2\lib\scriptgen.jar;C:\java_card_kit-2_1_2\lib\offcardverifier.jar;C:\java_card_kit-
 2_1_2\lib\api21.jar;C:\java_card_kit-
 2_1_2\lib\capdump.jar;c:\jdk1.3.1_01\samples\classes;%CLASSPATH%;

c:\jdk1.3.1_01\bin\javac -g -classpath %_CLASSES% %1.java %2 %3 %4 %5 %6 %7
```

5. 利用 build.bat 產生 java card applet class 檔，用法如下:
   build <filename>

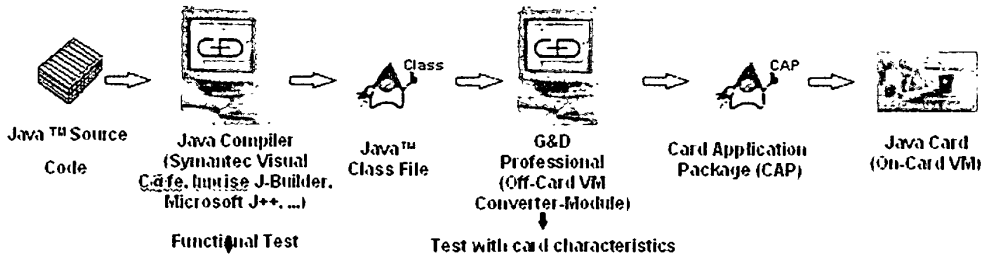
例如有一個檔名為 myfirst.java 的 java card applet 原始碼檔

build myfirst

將產生 build.class

(三) Sm@rtcafe 開發流程

首先要瞭解 Sm@rtcafe 開發 Java card Applet 的流程(如下圖)

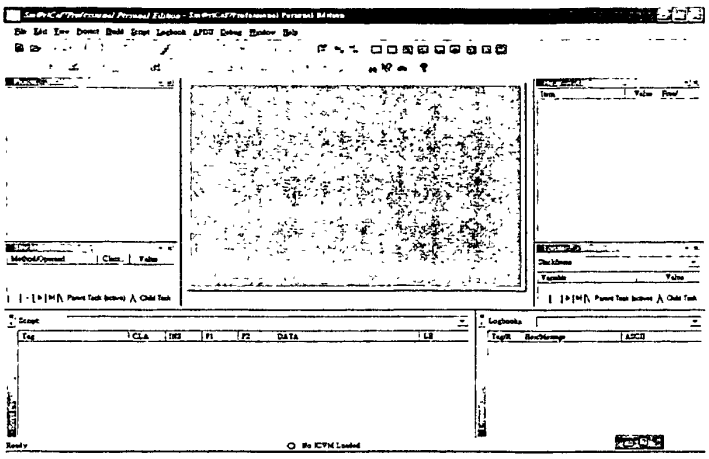


因應這樣的開發流程，Sm@rtcafe Project 有兩種 Card Template Project 模式(如下圖) 。

1. CardAcces 此模式為卡片實際存取運作情形
需搭配外接式讀卡機，並且插上 javacard 卡，而 javacard 本身也已載入在 Simulation 模式上測試完成的 java applet，載入(load) applet 到 javacard 需另外使用 load.exe 程式。
2. Simulation 此模式為開發 Javacard Applet，並模擬 Applet 在 Javacard 執行情形

因為 Javacard Applet 若已在 Simulation 模式下完成測試執行並且成功，在 CardAccess 模式下理應也能順利執行(除讀卡機或卡片有問題外)，所以將重點放在介紹 Simulation Project 的操作方式。也就是說明如何利用 sm@rtcafe 產生 Javacard Applet 及模擬 Applet 在 Javacard 執行情形。

1.執行 smart20.exe (檔案在安裝 sm@rtcafe 目錄下) ，剛開始之畫面如下。



2. 在 New Project 前，需要先準備好三個檔案，分別為.class、.opt 及.scr(以副檔名區別)。

.class：javcard applet 檔原始文字碼(.java)利用 build.bat 產生。

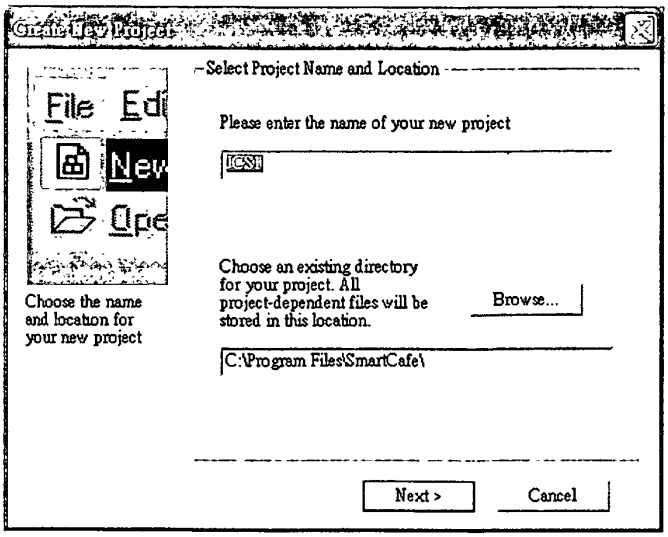
.opt 檔：javacard applet 檔案配置及 AID 宣告。

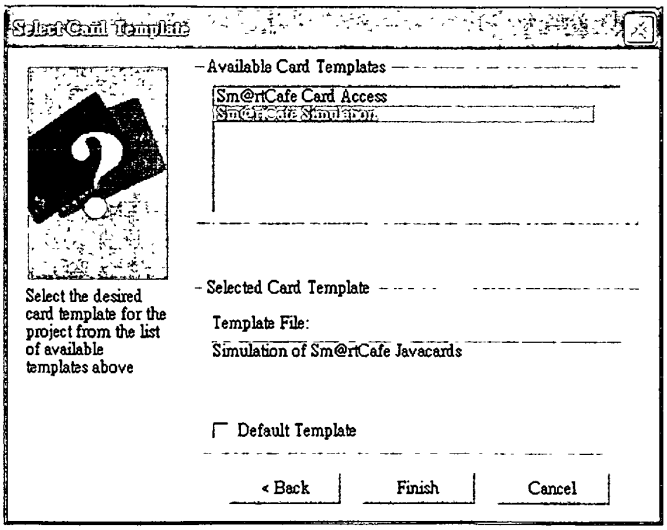
.scr 檔：apdu script，提供模擬 javcard apdu 操作。

可利用一般的文書編緝軟體來完成上述三個文字檔格式之檔案，也可在執行 New Project 後於 sm@rtcafe 中編緝三個檔案，視個人的操作習慣而定。

3.產生 New Project

利用滑鼠點選 File → New Project →輸入 Project 名稱及檔案位置→選擇 Sm@artcafe Simulation→按 Finsh 完成 Project。

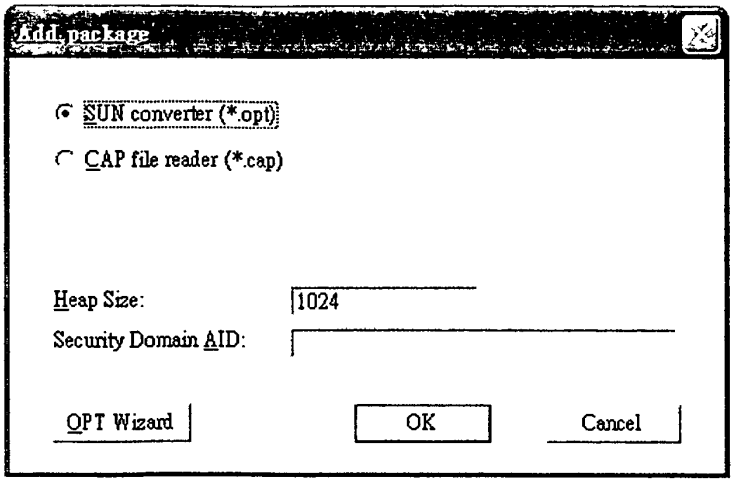




4.匯入 javacard applet 作業

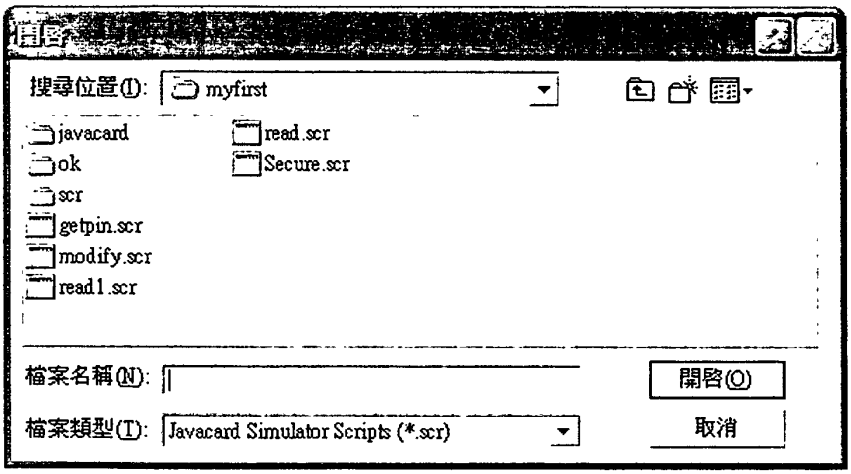
利用滑鼠點選 Project→AddPackage→選(*.opt)　在再點選 ok→選擇 opt 檔案

上述動作會依 opt 檔內容將 javacard applet class 匯入，若 opt 檔之內容宣告有錯誤，則 class 檔會無法匯入。



利用滑鼠點選 Script→Add script→選擇檔案(*.scr)

上述動作即可將 APDU script 匯入。



.class 檔，產生說明:

build <filename>

如有一 檔名為 myfilrst.java 之 javacrad applet 原始碼檔

build myfirst 將產生 myfirst.class 檔。

.opt 範例檔 格式如下：

-debug

-classdir c:\javawork1

-exportpath C:\Sm@rtcafe\Api

-out CAP JCA EXP

-applet 0x31:0x32:0x33:0x34:0x32:0x36 com.gieseckedevrient.applets.myfirst.MyFirst

com.gieseckedevrient.applets.myfirst

0x31:0x32:0x33:0x34:0x32:0xAA:0xFF:0xCA:0xFE:0xFF:0xAA 1.0

.scr 範例檔 格式如下：

:Install 80 E6 0C 00 1E 0B 31 32 33 34 32 AA FF CA FE FF AA 06 31 32 33 34 32 36 06 31 32 33 34 32 36 01 00 00 00 00

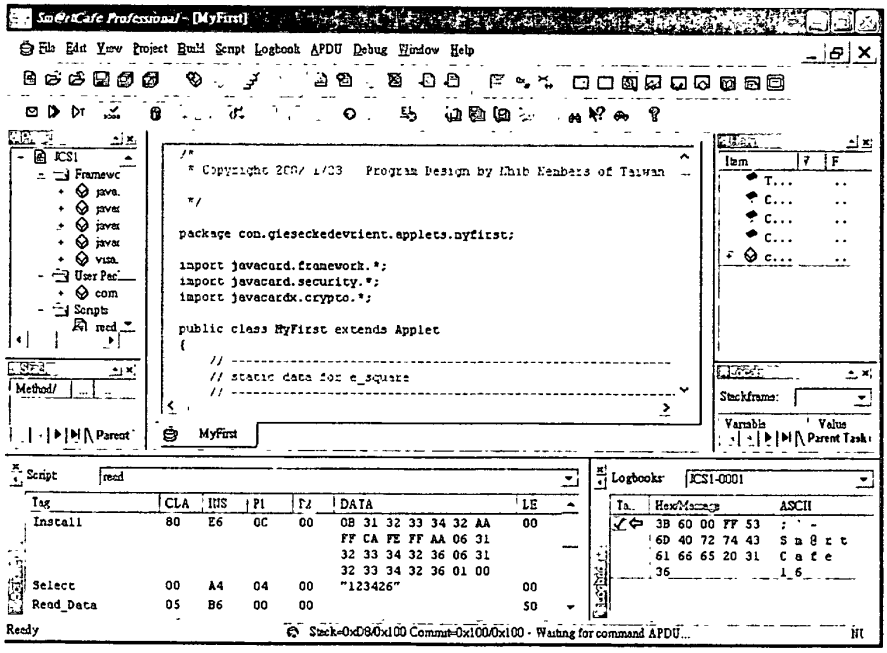
:Select 00 A4 04 00 06 "123426" 00

:Read_Data 05 B6 00 00 00 50

:Select_CM 00 A4 04 00 07 A0 00 00 00 03 00 00

:Delete 80 E4 00 00 08 4F 06 "123426"

完成匯入 javacard applet 作業後(如下圖)，即可進行模擬執行及實際利用 APDU script 測試模擬 javacard applet 的回應動作。



5.測試 java card applet 的模擬作業

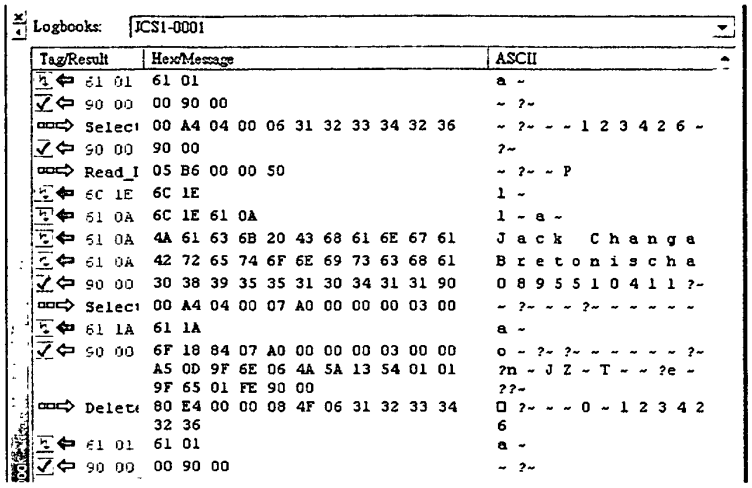
本作業目的就是當執行 java card applet，利用 APDU script 來確認回應結果是否正確。

首先利用滑鼠點選 Debug→InsertCard/poweron， 即卡片插入讀卡機並且啟動讀卡機電源的模擬作業。

再利用滑鼠點選 Script→Run script 即讀卡機上的 APDU 命令對卡片

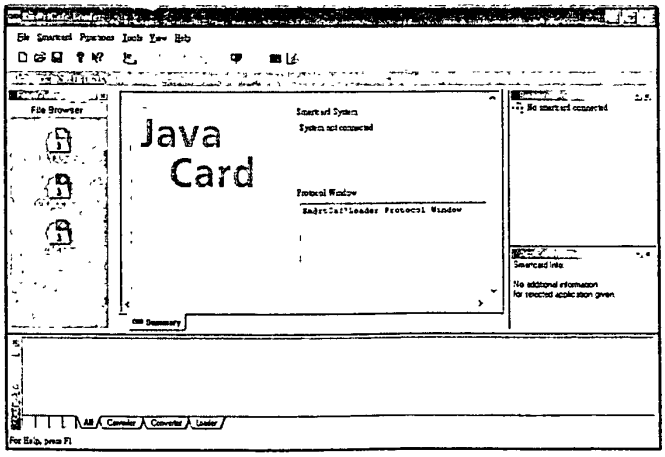
applet 進行測試的模擬作業。最後執行回應結果視窗的 logbook 中，若 APDU script 檔中有 "?"則會跳出一個小視窗，讓使用者可以輸入數值(一般是用來測試 pin 值) 。

下圖為 JavaCard Applet 的執行結果，顯示在 Logbook 視窗中。



6. java card applet 的實際作業

在各種 APDU script 模擬作業都成功後，接下來就是使用真實的 java card 及讀卡機來測試，當然首先要把 java card applet download 到卡片上，可以使用 sm@rtcafe load 的程式將 applet download 到卡片後，在利用 sm@rtcafe 開啟新的 Card Access Project 進行測試，其測試方式和步驟 5 相似，下圖為 load 的執行畫面。



7. 當完成 java card applet 實際作業測試後，就可以將載有 applet 的卡片插入讀卡機和開發完成的 Terminal application system 一起進行連線測試了。
8. 有關 sm@rtcafe 其他功能，可參考原廠 sm@rtcafe toolkit 使用手冊。

(四) 設計 Java Card Applet

1 Java Card Applet 簡介

1-1 什麼是 Java Card Applet?

Java Card 就是能夠執行 Java Program Language 的一種智慧卡（smart card），目前健保 IC 卡上所使用的 Java Card 版本為 Java Card 2.1.1 ，其規格說明可參考附件或網址 http://java.javasoft.com/javacard，而執行在 Java Card 上的 Java Program 稱為 Java Card Applet。

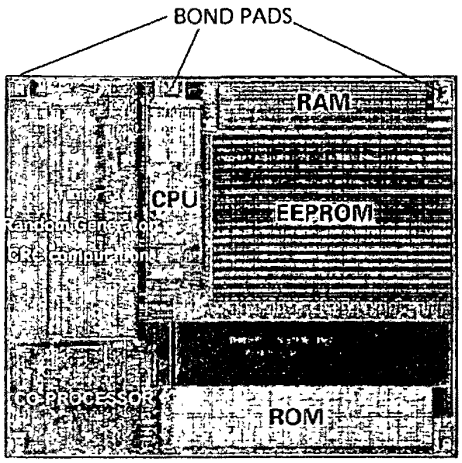
1-2 Java Card Applet 的執行環境

能讓 Java Card Applet 執行，其卡片上的系統晶片最低需求如下

- ✧ 16 kilobytes of read-only memory (ROM)
- ✧ 8 kilobytes of EEPROM
- ✧ 256 bytes of random access memory (RAM)

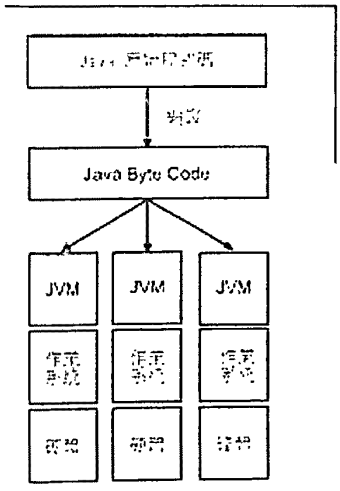
而下圖為型號:Sle66cx160S 的系統晶片硬體架構

- ROM: 32 KByte
- RAM: 1280 Byte
- EEPROM: 16 KByte
- CPU - 8 Bit
- I/O (Input/Output)
- 1MHz - 7,5MHz
- 21mm²



型號:Sle66cx160S

Java Card Applet 執行的環境稱為 JCRE (Java Card Runtime Environment) ，Java Card 執行環境需架構在 JVM 上，Java 程式並不像其它程式語言，最後被編譯成所在平台的機器語言後再執行，而是先編譯成一個中立的位元碼 (byte code)，然後才到裝有 Java 虛擬機器 (Java Virtual Machine， JVM)上的硬體去執行。下圖簡單的表示 Java 原始程式碼、位元碼、JVM 和硬體之間的關係。

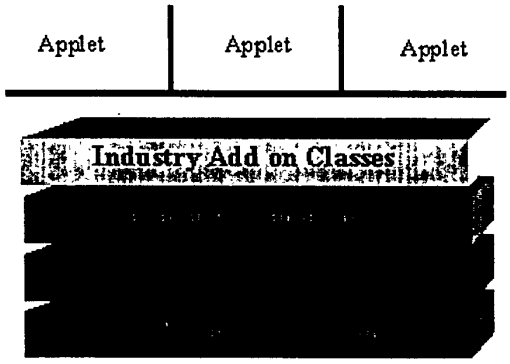


(一般 Java 架構圖)

JVM 目前已有多種平台的版本，像是 Windows、 Solaris、Linux、Macintosh 等...，除了這些較常見的作業系統外，還有針對各個小型的系統設計的 JVM，像是手機、PDA、Java Card 等，當然，健保 ic 卡 JVM 平台其就是 Java Card。

在整個 Java 的執行環境可以統稱為 JRE (Java Runtime Environment)，而 Java Card 執行環境(Java Card Runtime Environment)稱為 JCRE。JCRE 包括 Industry Add on Classes、 Javacard Framework、Javacard VM 等。

Java Card 架構如下圖所示



- Java Card VM(Java 虛擬機器簡稱 JVM) 被建造於特定的積體電路(ic)內及建置在作業系統之上，在 JVM 層包函了 java 技術所使用的通用語言和系統界面之直譯執行功能。
- Java Card framework 定義應用程式界面(API)中 Java Card 應用程式的開發物件及所提供的系統服務。
- Industry Add on Classes 為所新增特定的工業或商業物件程式庫，其主要為對另有特殊用途的系統模組重新定義及提供更具安全性的物件服務，其實就是為特定目的所新增加的 API。

◆ Java Card 應用程式稱為 applet，卡片上能存放多支的 applet . 每一支 applet 都有一個應用程式辨別碼 *AID* (application identifier)， 為 ISO 7816 第 5 部份所定義。

1-3 Java Card 程式語言的組成

Java Card 程式設計當然是用 Java 語言來寫，其編譯方式和一般的 Java compilers 是相同的，只是程式設計上會因侷限於有限的記憶體資源(limited memory resources)及計算能力(computing power)，所以並不是所有 Java Language 上所有定義的功能(如類別、物件、變數、界面及語法)皆能使用。 Java Card 不能支援功能如下：

- Dynamic class loading
- Security manager
- Threads and synchronization
- Object cloning
- Finalization
- Large primitive data types (float， double， long， and char)

4-1-4 Java Card API

Java 應用程式之所以能夠這麼簡單、快速的開發完成，而又能在各種不同的硬體平台上面執行，最大的功臣莫過於一堆為它量身訂做的 API 們，又可稱它們為類別函式庫。這些類別函式庫請參考 SUN 的網頁(http://java.sun.com/products/)，類別函式庫可分成四類。

1. 基本平台套件

   不管開發 Java 任何平台的程式，一定少不了這個基本的套件，它就是 JavaTM 2 Platform， Standard Edition，簡稱 J2SE。它除了包含開發 Java 程式所需的基本類別函式庫之外，還有一些編

譯的程式、額外的輔助工具等，開發 Java Card Apllet 也需要此套件來編譯。

2. 消費性產品及嵌入式系統套件

   有 PersonalJava、EmbeddedJava、Java CardTM、JavaPhone、Java TV 等，在開發 Java Card Apllet 時，就需使用 Java CardTM Platform。

3. J2SE 額外的輔助套件

   用來開發 3D 動畫、多媒體應用程式等。

4. 其它套件

   除了上面那些套件之外，還有一些其它的類別函式庫或是相關應用程式等，像是 JINI、JAIN、JavaTM Message Queue 等...。

目前健保 IC 卡上所使用的 Java Card API 版本為 Java CardTM 2.1.1 Platform，有關其詳細規格及作業說明可參考附件(Java Card 2.1.1 API Specification)或網頁 http://java.sun.com/products/javacard/javacard21.html 。

1-5 Java Card 2.1.1 API 套件

Smart Card 的應用在市場中已經有 20 年之久，其規格可參考(ISO 7816 ， part1-7)，而 Java Card API 套件(Package)是設計用來支援 Smart card 的系統及應用程式，此套件包藏著 Smart card 規格內各部分所應提供的細部功能，其實此套件就是提供一個容易的程式設計界面給 Java Card 系統程式開發者。

Java Card API 包括三個核心套件(core package)及一個延伸套件(extension package):

套件名稱	描述

Java.lang	This core package provides fundamental Java language support.
Javacard.framework	This core package provides framework classes and interfaces for the core functionality of a Java Card applet.Most impotant，it defines classes such as Applet and PIN，which are the fundamental building blocks for Java Card programs and APDU，System and Util，which provide runtime and system service to Java Card programs，such as APDU handling and object sharing
javacard.security	This core package provides a framework for the cryptographic functions support on the Java Card platform.It defines a key factory class KeyBuilder and Various interfaces that represent cryptogtaphic keys used in symmetric(DES) or asymmetric(DSA and RSA) algorithms.
Javacardx.crypto	This extension package defines the abstract base class Cipher for supporting encryption and decryption.

2 Java card Applet 程式設計觀念

開發應用系統軟體時，其步驟需進行可行性研究、需求分析 、系統開發、系統設計、程式設計、程式撰寫、程式測試、系統測試及系統安裝。

撰寫 Java Card applet 程式之前，首先需逐步進行 applet 程式架構的設計。其過程如下:

1. 依系統設計需求規劃 applet 整體架構及類別(class)細部功能(function)
2. 在設計 applet 及 package 時，要事先指定 AID 值
3. 定義 Applet 類別架構(class structure)及方法功能(method functions)
4. 定義 Applet 和 Terminal Application 的介面(interface)關係

接下來詳細說明設計步驟：

1.依系統設計需求規劃 applet 整體架構及類別(class)細部功能(function)

也就是一般傳統程式設計要做的事，依實際需求規劃程式架構及細部功能，如資料欄位及程式參數定義？class 之間資料的關聯如何？那些作業要在 class 集中處理、那些要分散在 class 處理？資料要如何傳遞？ class 具有那些功能？等等。

2.在設計 Applet 時，要先指定 AID 值

大部份所熟悉的應用程式命名方式都為字串格式(string name)，但在 Java Card 中，每個應用程式(applet)命名(identified)使用的是 AID 格式(**Application identifier**)。因此每個 Java package 均被指定一個 AID 值，這是因為 package 被載入到卡內後，會要讓其他的 package 來連

結（或呼叫），而它們放在卡內的辨別名稱所使用的格式已被定為 AID 值，其名稱轉換之規格同於的 ISO 7816 定義。

AID 是一組序列的位元組，其長度介於 5 到 16 bytes 間， AID 的定義及描述請參考下列表格

Application identifier (AID)	
National registered application provider (RID)	**Proprietary application identifier extension (PIX)**
5 bytes	0 to 11 bytes

AID 格式

ISO controls the assignment of RIDs to companies， with each company obtaining its own unique RID from the ISO. Companies manage assignment of PIXs for AIDs.

下表是自行舉例所定義 applet AID 格式說明：

Package AID		
Field	**Value**	**Length**
RID	0xF2， 0x34， 0x12， 0x34， 0x56	5 bytes
PIX	0x10， 0x00， 0x00	3 bytes
Applet AID		
Field	**Value**	**Length**
RID	0xF2， 0x34， 0x12， 0x34， 0x56	5 bytes
PIX	0x10， 0x00， 0x01	3 bytes

舉例實作 AID 格式

The package AID and the applet AID have the same RID value; their PIX values differ at the last bit.

3.定義 APPLET 類別架構(class structure)及方法功能(method functions)

Java Card applet 類別(class)必須繼承 javacard.framework.Applet 類別，這個物件是所有 applet 都一定會參考到，framework 是存在 Java Card 上的一個超級類別(superclass)，它定義一般 JCRE 執行週期時所支援 applet 共同的方法（common method）。

下列所示為 javacard.framework.Applet 中有關 public 和 protected 方法的類別定義：

Method summary

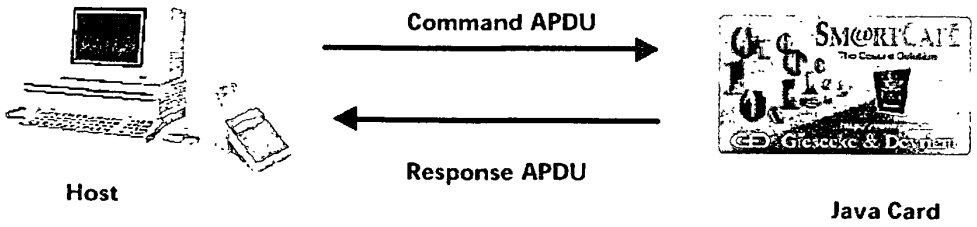
public void	deselect () Called by the JCRE to inform the currently selected applet that another (or the same) applet will be selected.
public Shareable	getShareableInterfaceObject (AID client AID， byte parameter) Called by the JCRE to obtain a sharable interface object from this server applet on behalf of a request from a client applet.
public static void	install (byte[] bArray， short bOffset， byte bLength) The JCRE calls this static method to create an instance of the Applet subclass.
public abstract void	process (APDU apdu) Called by the JCRE to process an incoming APDU command.
protected final void	register () This method is used by the applet to register this applet instance with the JCRE and assign the default AID in the CAD file to the applet instance.
protected final void	register (byte[] bArray， short bOffset， byte bLength) This method is used by the applet to register this applet instance with the JCRE and to assign the specified AID in the array bArray to the applet instance.
public boolean	select () Called by the JCRE to inform this applet that it has been selected.
protected final boolean	selectingApplet () This method is used by the applet process() method to distinguish the SELECT APDU command that selected this applet from all other SELECT APDU APDU commands that may relate to file or internal applet state selection.

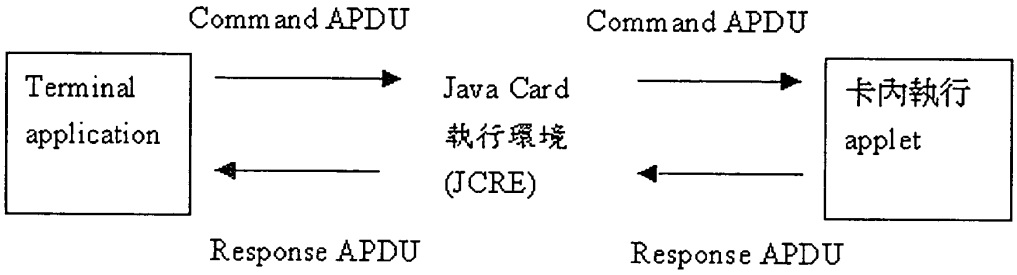
Public and protected methods defined in the class javacard.framework.Applet

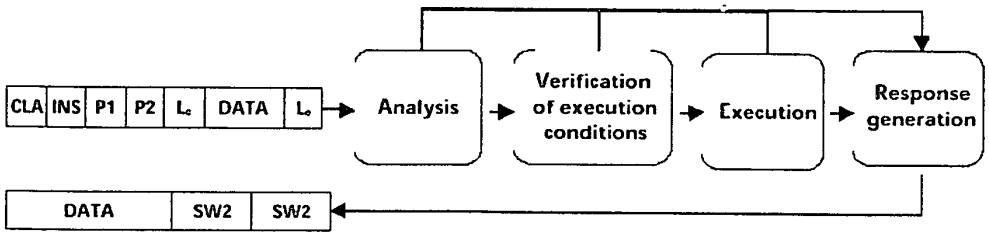
4.定義 Applet 和 Terminal Application 的介面(interface)關係

正在智慧卡中執行的 applet 是使用何種通訊協定(application protocol data units）跟讀卡機(CAD)上的端末應用程式進行資料交換動作，基本上 applet 和 terminal application 之間的通訊界面協定所使用的是，雙方都有支援的 APDU 命令集(set of APDU commands)，下圖表

示 Applet 和 Terminal Application 介面(interface)關係，兩者間的通訊協定為 APDU。







CLA	Class Byte	**SW1, SW2**	Status Bytes
INS	Instruction code	**Data***	optional
P1, P2	Parameters		
L	Command length		

APDU 說明

Command APDU

Mandatory header				**Optional body**		
CLA	INS	P1	P2	Lc	Data field	Le

- CLA (1 byte): Class of instruction --- indicates the structure and format for a category of command and response APDUs
- INS (1 byte): Instruction code: specifies the instruction of the command
- P1 (1 byte) and P2 (1 byte): Instruction parameters -- further provide qualifications to the instruction
- Lc (1 byte): Number of bytes present in the data field of the command
- Data field (bytes equal to the value of Lc): A sequence of bytes in the data field of the command
- Le (1 byte): Maximum of bytes expected in the data field of the response to the command

Response APDU

Optional body	**Mandatory trailer**	
Data field	SW1	SW2

- Data field (variable length): A sequence of bytes received in the data field of the response
- SW1 (1 byte) and SW2 (1 byte): Status words -- denote the processing state in the card

Command 和 response APDU 格式

Applet 在處理 APDU 命令時，所利用到的 APDU 物件之方法（method）及步驟詳述如下：

步驟 1. Retrieve the APDU buffer

The applet invokes the ***GETBUFFER()*** method to obtain a reference to the APDU buffer， which contains the message. When the applet receives the APDU object， only the first five APDU header bytes are available in the APDU buffer. They are the CLA， INS， P1， P2， and P3 bytes respectively. Byte P3 denotes the Lc byte， if the command has optional

data. The applet can inspect the header bytes to determine the structure of the command and the instruction specified in the command.

步驟 **2. Receive data**

If the command APDU contains optional data，the applet must direct the APDU object to receive incoming data by invoking the *SETINCOMINGANDRECEIVE()* method. The data is read into the APDU buffer following the five header bytes. The last byte in the header (Lc) shows the length of the incoming data. If the APDU buffer can't hold all the data，the applet can process the data piecemeal，or it can copy it to an internal buffer. In either case，it would then repeatedly call the *RECEIVEBYTES()* method to read additional data into the APDU buffer.

步驟 **3. Return data**

After processing the command APDU，the applet can also return data to the CAD in the response APDU. The applet should first call the *SETOUTGOING()* method to set the data transfer direction to outbound，and to obtain the expected length of response (Le). Le is specified in the command APDU paired with this response APDU.

Next，the applet calls the *SETOUTGOINGLENGTH()* method to inform the CAD of the actual length of the response data. The applet can move the data to the APDU buffer and call the *SENDBYTES()* method to send out data. The *SENDBYTES()* method can be invoked repeatedly if the APDU buffer cannot hold the entire response data.

If the data is stored in an internal buffer，the applet invokes the *SENDBYTELONG()* method to send data from the buffer.

If the response data is short enough to fit into the APDU buffer，the *APDU* class provides a convenient method for doing so: *SETOUTGOINGANDSEND()*. This method is a combination of *SETOUTGOING*，*SETOUTGOINGLENGTH*，and *SENDBYTES*. However，this

method can only be invoked once， and no other send methods can be invoked afterwards.

步驟 **4. Return status word**

Upon a successful return from the ***PROCESS()*** method， the JCRE automatically sends ***0X9000*** to indicate normal processing. At any point， if the applet detects any error， the applet can throw an ***ISOEXCEPTION*** by invoking the static method ***ISOEXCEPTION.THROWIT(SHORT REASON)***. The status word is specified in the parameter ***REASON***. If the ***ISOEXCEPTION*** isn't handled by the applet， it will be caught by the JCRE. The JCRE retrieves the ***REASON*** code and sends it as the status word.

三、*Java Card Applet 測試*

(一)INQSmart 軟體簡介

INQSmart 整合式測試軟體是由比利時 integri 公司所開發(註)，它是架構在 INQ(同樣為該公司開發的一套整合工具)平台上，並且涵蓋了一些有用的工具軟體，可以用來測試及模擬智慧卡和智慧卡讀卡裝置。

正因為智慧卡上嵌入(相對而言)昂貴而精密的微處理器(microprocessor)，以及不易修改的軟體，使得它的製造過程變得比一般產品更加困難，同時因為智慧卡一旦開始發送和安裝後極可能難以回收，因此發卡單位必須先完成必要的測試，以確保其品質及降低未來可能的回收成本。讀卡裝置的廠商也可能面臨到以上的問題。

雖然目前已經有國際標準(ISO/IEC 10373， EN1292)及相關工具可供測試有關卡片的各項物理和實體係數，但是仍然需要一套適合的方式來測試驅動程式(driver)和軟體，尤其智慧卡上的軟體都是用高達數千行的組合語言或 Java 撰寫而成，要如何有效地測試各項細節(例如 mask 和個人化)顯然必須好好考量。

根據原廠的簡介資料，這套工具的優點至少包括：

- 自動化：可啟動多個自動排程測試，減少人工執行及測試時間。
- 功能完整：可自行撰寫 test scripts 作各種狀態的測試
- 可重覆執行：一旦增加功能或出新版軟體時，可用最少的成本重新測試。
- 有效節省人力：自動執行繁瑣程序有助於將測試人員時間花在更有意義的錯誤分析上。
- 各種執行模式：提供測試模式、模擬模式和除錯模式因應不同需

求，並且可以在不同模式間快速切換。

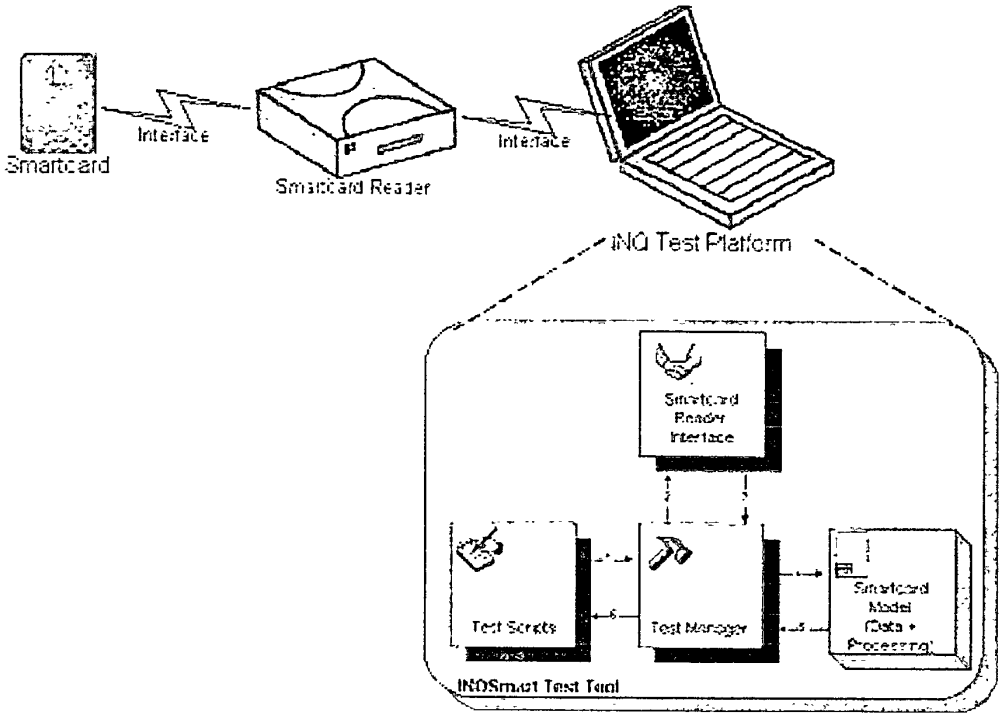
- 整合性：所有的工具都是架構在同樣的設計理念和使用者介面，彼此之間溝通容易。
- 縮短專案開發時程：一般的智慧卡系統要作測試，必須先取得實體的卡片。但是可透過本軟體的模擬程式提早測試作業；同時它也不必非用到真正的終端機，只要安裝簡易的讀卡機，再以自動化工具程式讀取卡片即可。

INQSmart 其實包含了 INQSmart Test Tool、INQSmart Emulator、INQSmart PassThrough、INQSmart Logger 及 INQSmart Level1 Replayer 等不同模組，針對 smartcard 的各個硬體或軟體部分，分別定義測試細目，所有的模組則集合成一套完整的測試軟體，使用者則可視實際需求加以選用。

其中又以 INQSmart Test Tool 較為常用，它主要針對智慧卡上的應用程式執行測試作業，它可以對卡片作業系統(Card Operating System)、嵌入式系統、個人化、狀態等作完整的檢驗，若配合讀卡裝置使用亦可作其他進一步的測試。

簡單來說，本章所要介紹的測試工具 INQSmart 和前面一章提到的 SM@RTCafe 最大的差別在 SM@RTCafe 可以開發將來實際可下載到卡片上運作的 APPLET 程式，而 INQSmart 則主要是用來測試寫好的程式及開發模擬程式，換言之寫好的程式只能在個人電腦的模擬環境下執行，不能下載到卡片上。

INQSmart 的運作示意圖：



(二)測試內容說明

以這次由德國 G&D 公司所開發的卡片程式(Java card applet)為例，在最後的測試階段即選用 INQSmart 相關工具軟體，茲說明如下：

整個健保 IC 卡 applet 的測試過程大致可分為：

1. 個人化(personalization)測試：包含啟動(initial)及執行(startup)。

例：基本的寫入及讀取卡片測試，及植入 key 的測試。

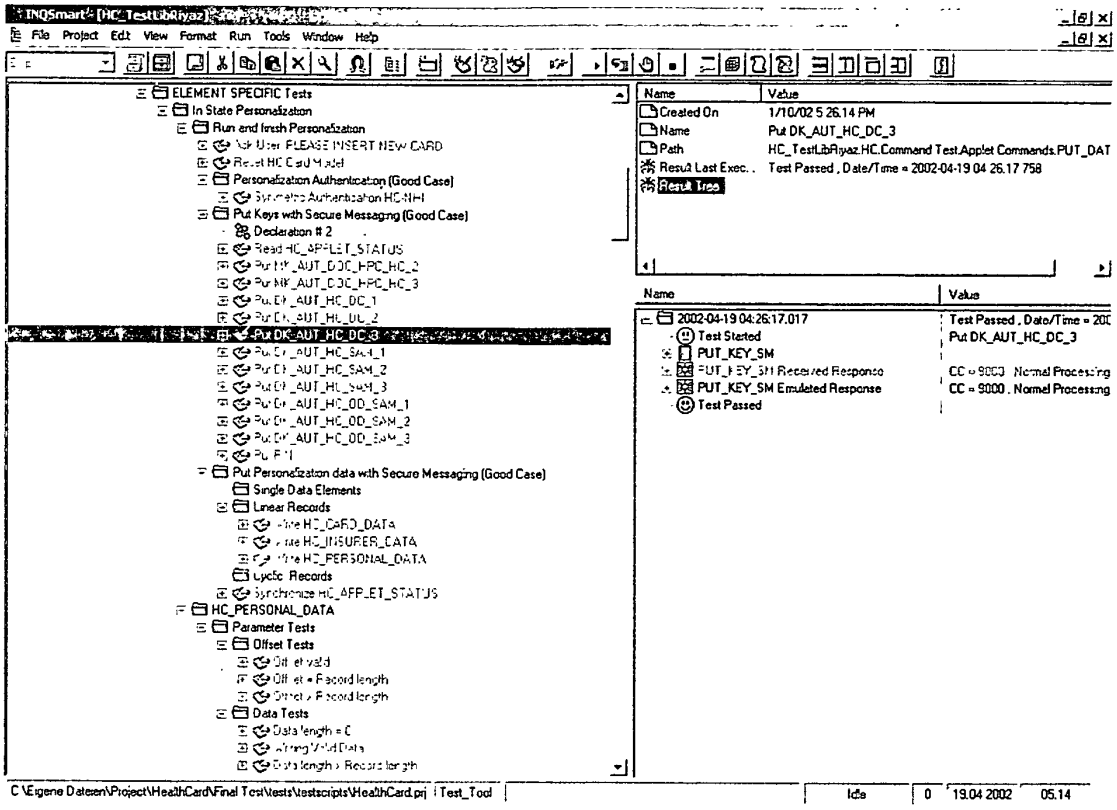
2. 狀態機(state machines)測試：依據事先作好的狀態分析，測試從不同狀態可否到達另一狀態。

例：假設規格書定義從 state 1 不可能下一步直接跳到 state 3，就要測試是否存在任何可能性導致此情形發生。

3. 指令(commands)測試：將預先設計的不同指令送到不同狀態，確認得到應產生的回應內容，同時不會產生非預期的副作用(side effect)。

以上的三種主題測試中，又以指令測試的工程較為複雜，所建立的測試樹(testing tree)也較為龐大。所謂的測試樹是指類似在 windows 作業系統中我們常用的檔案總管一樣，將所有的待測項目依照其從屬關係先區分為大分類，大分類以下再分成若干中分類，每個中分類下再分成幾個小分類，最下一層才是細目的方式排列；測試過程則是依序測試所有細目，一旦某個小分類下的所有細目都已通過測試，則此小分類即視為完成測試，一旦某個中分類下屬的小分類完成測試，則該中分類即算是測試通過，依此類推，一旦該測試樹下的所有細目測試完畢，即完成全部測試作業。

測試樹的架構如下圖：


File Project Edit View Format Run Tools Window Help
ELEMENT SPECIFIC Tests
In State Personalization
Run and fresh Personalization
Personalization Authentication (Good Case)
Put Keys with Secure Messaging (Good Case)
Declaration # 2
Put DK_AUT_HC_DC_3
Put Personalization data with Secure Messaging (Good Case)
Single Data Elements
Linear Records
Cyclic Records
HC_PERSONAL_DATA
Parameter Tests
Offset Tests
Data Tests
Name
Value
Created On
1/10/02 5 26.14 PM
Name
Put DK_AUT_HC_DC_3
Path
HC_TestLibRiyaz.HC.Command Test.Applet Commands.PUT_DAT
Result Tree
2002-04-19 04:26:17.017
Test Started
PUT_KEY_SM
PUT_KEY_SM Received Response
PUT_KEY_SM Emulated Response
Test Passed
Test Passed , Date/Time = 200
Put DK_AUT_HC_DC_3
CC = 9000 Normal Processing
CC = 9000 , Normal Processing
C:\Eigene Dateien\Project\HealthCard\Final Test\tests\testscripts\HealthCard.prj
Test_Tool
Idle
0
19.04.2002
05.14

因為指令測試是針對撰寫好的 applet function 作測試，它的測試內容項目又特別加以細分以下幾項：

1. 狀態(status)：測試狀態反應是否如預期。
2. 參數(parameter)：將參數用不合理的長度、個數送入該指令，看是否會產生應有的錯誤。

   以 APDU 指令為例，包括了前提(precondition)、一般性(general)、P1、P2、Lc、Data、Le 等不同 fields 都要一一測試，如果該指令含有特定參數(element specific)時也不可省略。
3. 資料(data)：設計特定的參數值傳入，確認可正常辨認及處理。
4. 整體訊息：確認所有的訊息代碼都可以產生。

   例如：所有的訊息代碼共 100 組，要測試每一組代碼都一定可以在某種情況下發生，不會有永遠用不到的代碼。
5. 隨機差異(random differences)：
6. 特定測試：依指令不同特有的反應。

   例如：可能大部分的指令都是只傳回 ok 或 error，但 GET_CHALLENGED 卻是去取得一個系統亂數，亦即會 response data 的指令，就必須測試傳回值是否合理。

以上簡單說明了測試的項目之後，接下來說明如何進行測試。

就如同資訊界有句話：『世界上不存在百分之百完美的程式』一樣，同樣在一個專案中，也沒有任何專業團隊或任何工具軟體，可以作到百分之百保證沒有 bugs 的測試，所以只能在人力、物力、成本、時間和項目內容上取一個平衡點進行測試，以德國 secartis 公司的作法為例，他們是按照參與人員的階級及職務，規劃四層式的測試步驟：

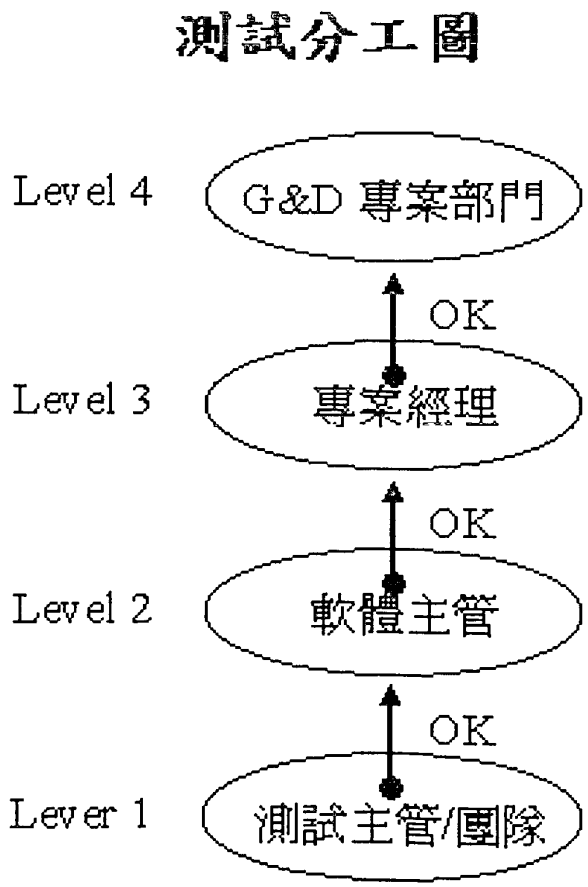
第一層由該專案之測試團隊的主管帶領測試小組成員設計所有的測試 cases，並進行所有的測試，這部分也是時間最久，花費人力最多的測試，下一段會再詳細說明。

第二層由該專案的軟體應用程式主管(帶領程式開發及測試二個團隊)本身執行選擇性、隨機性的測試。

第三層由該專案的專案經理(負責軟體及其他硬體支援項目)本身執行選擇性、隨機性的測試。

以上均完成後，就進入第四層，送交母公司的專案控管部門的專業人員作最後的測試及確認。

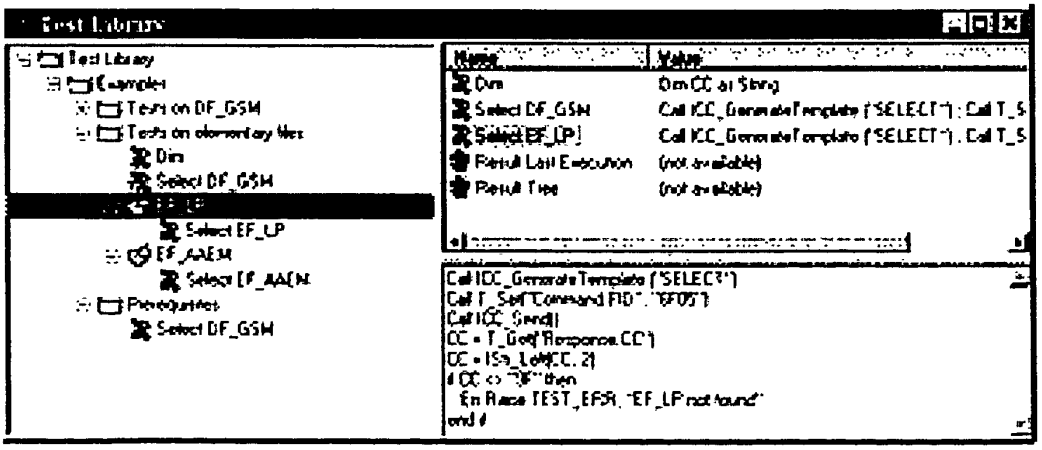
如下圖所示：



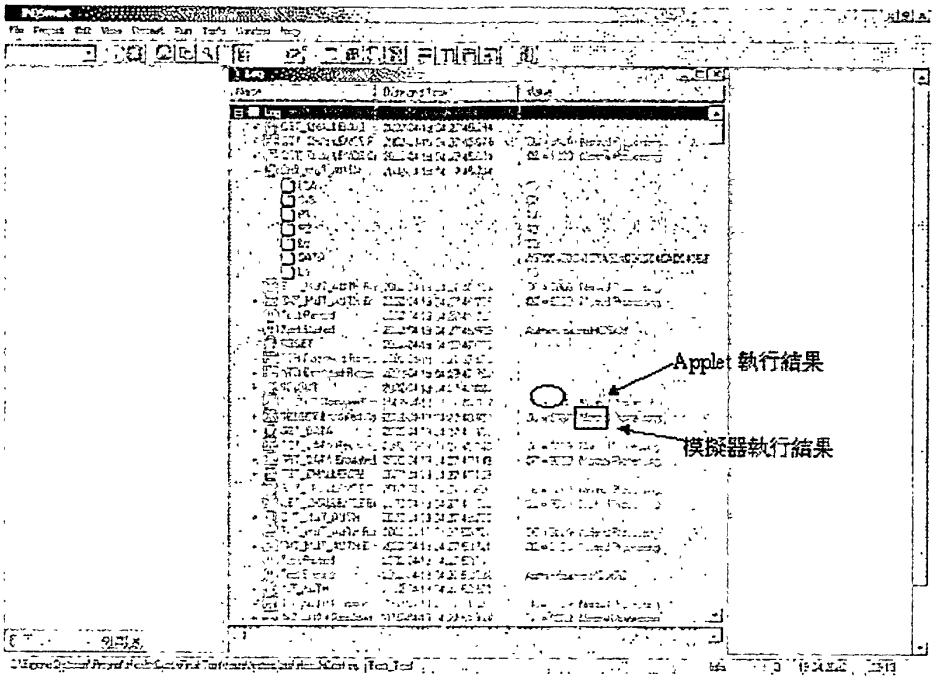
這樣作最大的目的在兼顧相關人員權責，經由不同人員的參與，減少遺漏測試項目的可能性，因為第二層到第四層的測試項目都是由測試人在測試時才依照自己的時程隨機選擇若干項目，並不是直接使用別人的測試項目，理論上也比較不會受到別人的測試結果所影響。

當然，這其中又以第一層的測試為主要核心，在這個階段 G&D 的工程師採用了一套國內很少見的工作模式，簡單說明如下：

首先，全體測試團隊先研讀系統分析師提出的系統規格書，確認已了解程式需求及功能，由其中一人設計所有的測試個案，交由測試團隊主管審核確認，再將其餘人員分成二組，一組負責用 INQSmart 直接測試程式設計師寫出來的 Java applet 程式，另一組則利用 INQSmart 本身支援的模擬器(emulator)功能撰寫測試用 script(語法類似 visual BASIC)，script 的內容如下圖右下角：



這二組的成員必須是獨立的，也就是不會有人同時屬於這二組，組內同仁可以盡量交換意見，但組和組之間要減低互相影響的可能性。最後再比對這二組的測試結果內容，如下圖：



綠色(圓圈)的文字表示實際測試 applet 的結果訊息，黑色(方框)的文字則表示模擬器執行 script 後的結果，二者以上下列並排的方式呈現，以利於比較。按照正規的測試方式，所有的測試程序(testing procedures)都須經過比對綠色和黑色的執行結果，確認訊息完全一致後才通過測試，若是在其中任一項有不一致的結果，就必須另外挑出來重新分析，以得到正確的數據。這樣做的用意是為了降低程式設計師誤解原始需求的可能性。換言之，只有在 applet 程式開發人員和 script 開發人員同時發生同樣的錯誤時，才有可能產生未檢查到的程式錯誤，但其機率應是相當低了。

對照過去的測試經驗，這種作法雖然會增加測試成本，(事實上該公司才剛採用這種測試方式不久)，但是卻可有效減少日後出錯的

機會，特別是針對這類智慧卡系統所開發的程式，一旦發生錯誤後，所必須花費在補救的成本(例如重新燒錄卡上程式)極可能遠大於程式開發階段測試的成本，因此從長遠的角度來看，這樣做仍然是划算的。

以這次健保ic卡的applet程式來看,所有的測試程序保守估計達5000個以上,secartis共用了7個測試人員(其中3人為part-time)，費時3個月才完成第一層的測試工作。

至於如何確保所有測試個案無一遺漏?本專案採取的是

data element access security 的策略,也就是畫一個類似棋盤的格子,將所有狀態和命令分別列在橫軸和縱軸,每測試一個 ok,即在對應的交會格上打勾,直到所有的方格都已打勾才完成所有個案測試.

如下圖所示：

States/command	State 1	State 2	State 3	....
Command 1	V	V		
Command 2		V	V	
Command 3		V		

.....

.....

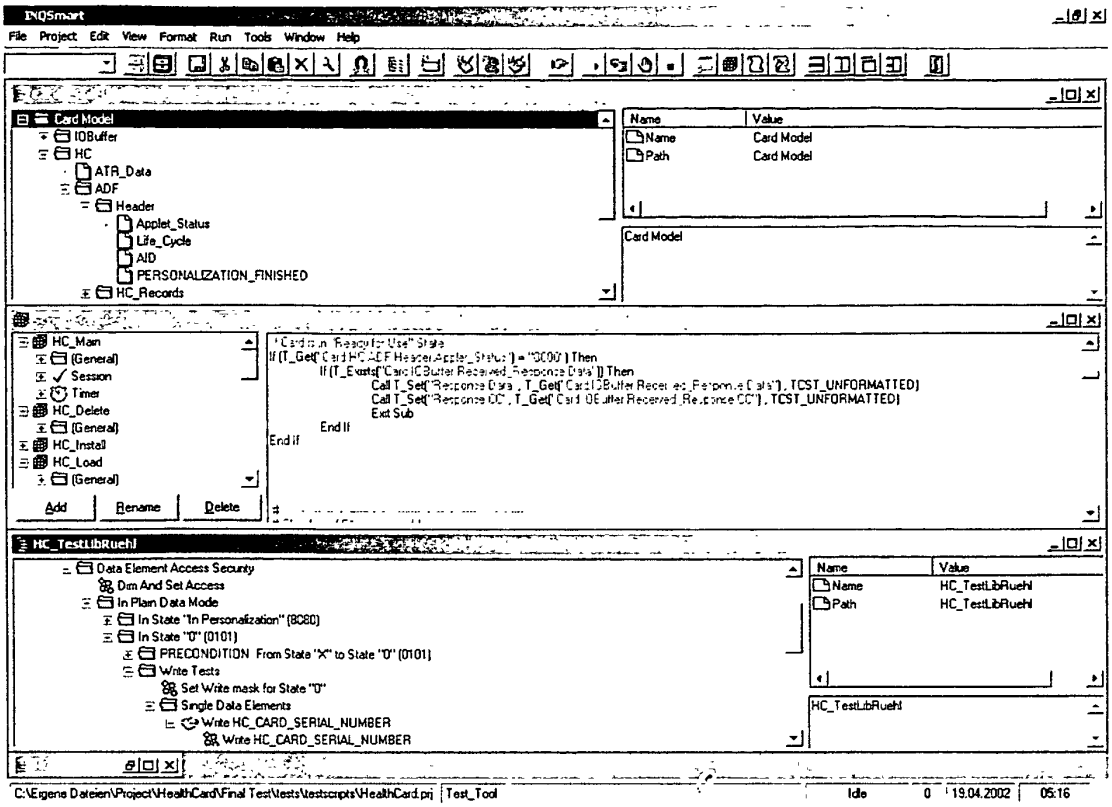
特別要加以說明的是,在整個 ic 卡專案中,金鑰(key)的測試和其他功能的測試內容稍有不同,因其純粹是作內部運算,並沒有外部使用者可查知的明顯訊息,但又因為它是整個安全機制的核心,所以也另外設計了一些必要的測試項目,包括

1. key derived test
2. key generation change
3. key access security
4. key fault presentation counter test..

等等..

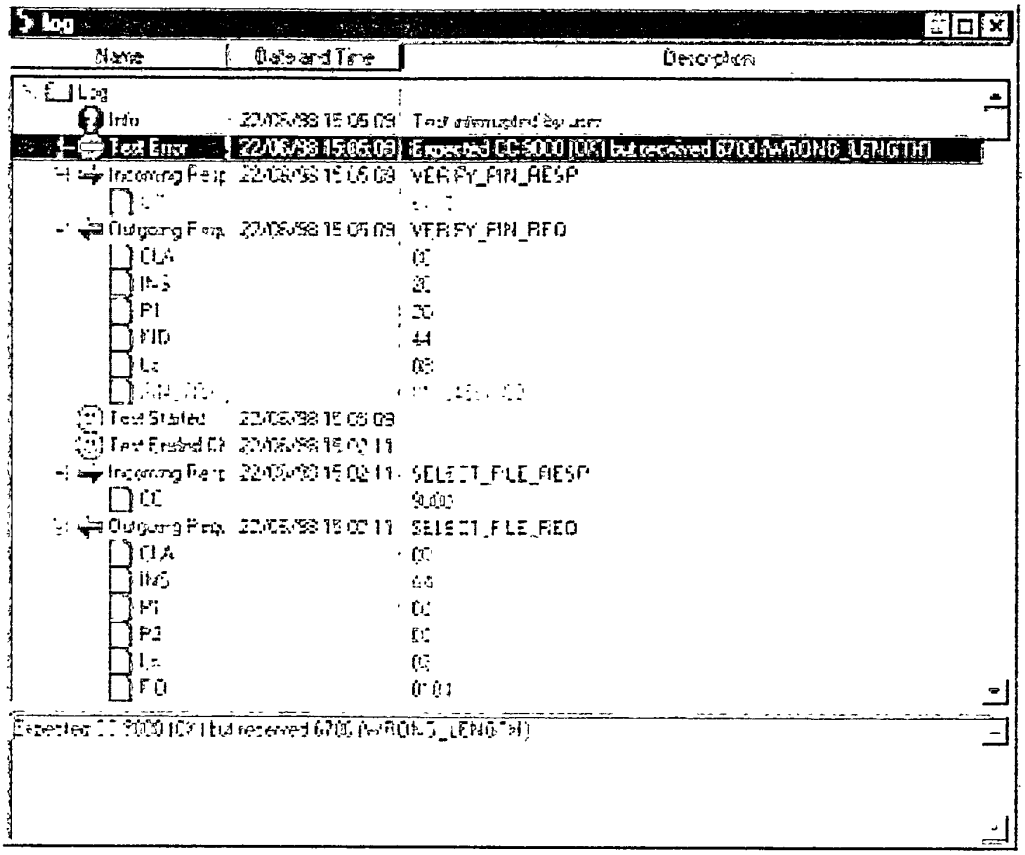
以 key fault presentation counter test 的內容為例，假定需求規格書規定 key 認證超過 20 次未成功就視為認證失敗，或是 pin 連續輸入 3 次錯誤就要將卡片鎖住，則在測試時就必須實際連續執行指定的次數，以確定內部的計數器正常運作。

透過專業的測試工具，可以記錄所有參與測試人員的測試內容，分別儲存成為不同的測試樹，並可在一個畫面下的不同的視窗同時呈現出來，如下圖：



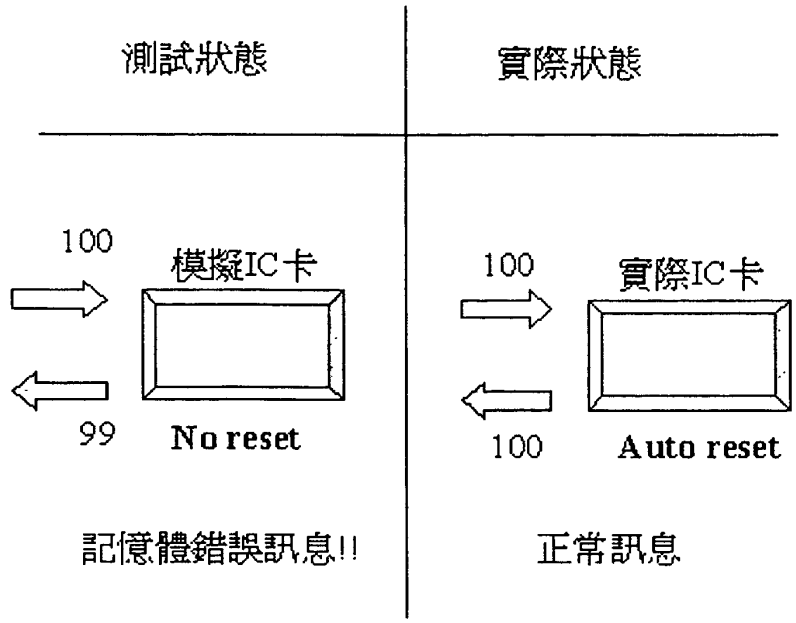
這樣做可以讓每一個測試人員參考別人的測試內容和結果，在節省時間和完整性的考量下，之前做過的測試可以不必重覆。

經過不斷的測試，修正，再測試，以達到一個所謂"bug-free"的成品，而最後所有的測試過程 log 都應該保存下來，以作為未來再次修正或是重覆確認的參考。下圖則是一個 log 的畫面：



有了較佳的測試工具軟體作為輔助，就可能突破現實環境的限制。例如我們的測試環境雖然只有一張 IC 卡，但可以用軟體去模擬醫事人員卡或安全模組卡的內容，以測試多張卡片間的互動狀況；或是用軟體加解密的方式加上簡單的讀卡機去模擬實際所採用的含有安全模組的讀卡機。這樣做也可以讓測試工作不需要等待實體產品開發完成。

但是測試畢竟還是和現實的環境不完全相同，測試工具軟體呈現出來的數據並不會百分之一百等同於最後的系統運作結果，以一個簡單的例子為例：



假定我們在測試環境下作卡片讀寫動作的測試，由於軟體本身特有的限制，會產生記憶體空間垃圾(garbage space)累積現象，讀寫前假定有 100 個可用的記憶空間(available memory)，讀寫後只會剩下 99 個，如此經過重覆多次的讀寫測試後，會產生記憶體錯誤訊息；反之這種情形在真實環境下非常不容易發生，因為卡片在作實際插入取出的讀寫時，因為電流訊號的改變，硬體機制會自動作記憶體重置(reset)，就不會有垃圾空間的存在。

按照 secartis 公司過去的經驗，程式開發設計和測試所各花費的人力時間約是 4 比 6，也就是說一個系統花在測試上的成本還要超過程式設計部分；測試的重要性在於雖然設計過程中也會夾雜有必要的測試，但通常只限於正常流程(good cases)；而專業的測試團隊則是要嘗試所有的可能，包括正常流程和異常流程(bad cases)，以確定不會發生不該發生的事。用一個簡單的觀念來說明，程式設計的人負責一加一等於二，可是完整的測試要確保不會有二加一等於二，或是

在某種情況下一加一等於三！

當然由於時程和分工的關係，這次前往德國的時間有大部分的時間是在了解和學習程式設計，並沒有看到太多的測試過程（有些程式尚未完全設計完成，更不要說測試了），然單就看到的部分已經值得未來多加參考。

註：報告內容部分文字資料或圖片取材或翻譯自 integri 公司原廠網站(www.integri.com)

參、 心得

一、持平而論，Java card applet 的撰寫並不算特別困難，一旦確定功能需求，只要先從幾個簡單的範例程式著手，並備好 Java card API 或 Open Platform 的規格以備隨時翻查，應該就能作簡單修改。但是依照國外的經驗，真正的學問是在"最佳化"調整的過程。如果把智慧卡視為一台微型電腦，相對於一般個人電腦的擴充性高成本低廉，智慧卡上的記憶體則相當有限，CPU 速度也多半只有 8-bit 的情況下，如何將程式碼變得更小，更有效率，執行速度更快(比如說能呼叫標準的 Java card 標準 package 做到的，儘量避免自行撰寫，否則會使得程式碼變大，執行時間也會相對延長)，減少卡片使用記憶空間(須配合資料欄位架構設計一併納入考量)，進而加速整體卡片讀寫時間，顯然就需要經驗累積，甚至需要經過嘗試比較不同作法後才能定案。

二、軟體工程的書籍常提到測試作業佔整個系統開發成本的比例相當高，或許不一定適用在小型而單純的資訊系統；但以這次和德國工程師討論所交換心得，針對像智慧卡這樣的系統，由於考慮到後續製卡、發卡等較傳統應用軟體複雜的情況，為了儘可能減少出錯重來的機會，測試作業花費的時間有可能高達整個系統開發時程的一半以上，但是這樣的代價應當是必要的。

三、有許多國內資訊廠商在開發資訊系統的過程中，基於成本及開發時程的考量，往往由程式設計人員充當測試人員，甚至還要負責撰寫系統文件；在沒有良好的品管下，往往因為設計者先入為主的觀念而隨便加以測試，或是寫出來的說明手冊難以理解。但以本次造訪的德國公司為例，就明確的將這三者的職責加以區分，避免單一人員過重負擔。其實只要內部有合理控管和分工，從顧客的角度來看，反而更能提高產品品質，長期而言也有助於減低成本。

四、本次出國期間也順道參觀了德國當地的印紙廠和製卡廠，並由廠方人員陪同說明，雖然限於時間和安全的考量，並無法完整看到所有的過程，但從一些諸如空間規劃、工作人員管理、流程控管、軟硬體設備的維護、乃至各項安全措施上，均可從中感受到德國人一絲不苟，實事求是的精神和效率，足可供國內借鏡。

肆、 建議

健保 IC 卡計畫目前正如火如荼的進行中，一些相關的應用系統依照時程也已將進入最後的測試或建置階段，茲建議數點如下：

一、基於未來仍有可能配合政策而變更需求，有關這次德國公司所使用的程式開發及測試工具軟體，建議可導入本專案作為未來自行測試或修改程式使用，或是另外引進類似的工具軟體，以建立一套完善的開發測試環境，並需視情況培養本土種子人員。

二、IC 卡計畫規模之大，影響層面包含行政業務，法律，資訊技術面等相關問題，單以資訊系統而言，依照功能不同可再細分為數十個子系統，各個子系統內容雖規模有大有小，但其重要性均無庸置疑，不可偏廢，而其開發過程除了必須確保單一子系統的正常運作外，還需一併考量系統之間有無良好的連貫性，因此必須建立共同遵循的設計介面和標準，以減少日後維護的不便。

三、在健保 IC 卡上路後，如想再規劃卡片所餘空間開發其他用途，如電子錢包等應用，務須及早準備，並要注意與原系統的相容性。

四、政府不惜鉅資，致力推動健保 IC 卡換發計畫，為多年來民眾普遍認同的全民健康保險制度帶來重大變革，著眼點在提昇我國的醫療產業和資訊產業，可謂用心良苦。因此一切有待參與的本局同仁、東元公司、社會大眾、醫療院所及相關資訊廠商多方配合及溝通，方能使計劃順利成功。

伍、 附錄

Java card applet 範例

以下三個 Java card applet 範例程式是在德國學習時的練習作業，可以用來幫助了如何應用 Java card API，進而掌握程式的設計技巧。

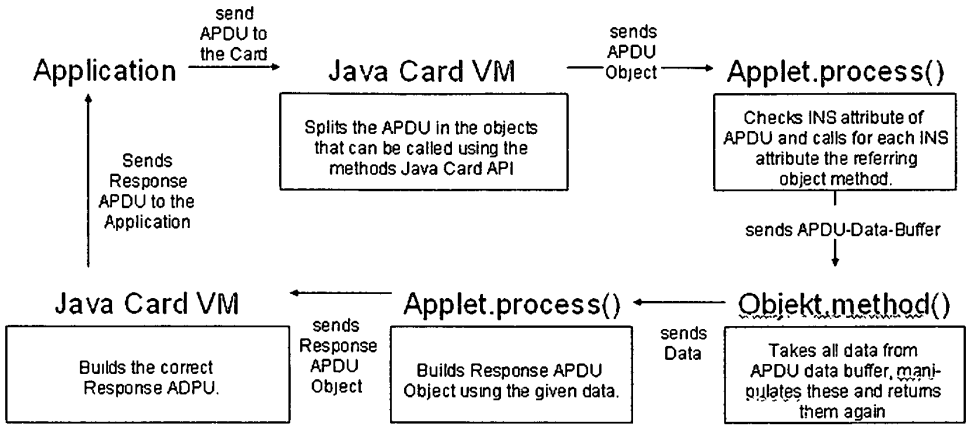
範例一　資料讀取

目的：

設計能夠讓讀卡機讀取卡片內資料功能的 applet，資料欄位有姓名(name)，電子郵件信箱地址(email address)及電話(Tel)。

說明：

本範例將能瞭解 applet 在處理 APDU 命令時，所利用到 javacard.framework的 APDU 相關物件之方法、函數的設計方式，下圖為 APDU 處理流程。




```
/**
    Exec1 :Sample Frame for JavaCard development
          Program Design by Nhib Nembers of Taiwan
*/
package com.secartis.applets.Frame;
import javacard.framework.*;
public class Frame extends Applet
{
   final static byte   NONISO_CLA              = (byte)0x00;              //CLA
   final static byte   READ                    = (byte)0xB6;              //INS
   final static byte   WRITE                   = (byte)0xD6;              //INS
   final static short        SW_INSTRUCTION_NOT_SUPPORTED = (short)0x6D00; //StatusWord
   private byte[] echoBytes;
   private static final short LENGTH_ECHO_BYTES = 60;
```

```
    private short echoOffset ;
    short total_bytes =(short)60;
private byte[] nameBytes;
private static final short LENGTH_NAME_BYTES = 8;
private byte[] emailBytes;
private static final short LENGTH_EMAIL_BYTES = 40;
private byte[] telBytes;
private static final short LENGTH_TEL_BYTES = 12;
public static void install(byte[] buffer ， short offset ， byte length)
{
  new Frame();
} // 'Applet.install'
  /**
 * Constructor.<BR>
 */
public Frame()
{
  echoBytes = new byte[LENGTH_ECHO_BYTES];
  nameBytes = new byte[LENGTH_NAME_BYTES];
  emailBytes = new byte[LENGTH_EMAIL_BYTES];
  telBytes = new byte[LENGTH_TEL_BYTES];
  register();
} //
/**   process method basic method to handle card commands    */
public void process(APDU apdu)
throws ISOException
{
  if( selectingApplet() )
       return;
  // get the reference to the APDU Buffer
  byte[] buffer = apdu.getBuffer();
  switch(buffer[ISO7816.OFFSET_CLA])
  {
  case NONISO_CLA:
    switch (buffer[ISO7816.OFFSET_INS])
    {
      case READ:
  //      echoBytes = (byte)(nameBytes + emailBytes + telBytes );
```

```
            apdu.setOutgoing();
            apdu.setOutgoingLength( total_bytes );
            // echo header
            //apdu.sendBytes( (short)0 , (short) 5);
            // send the name
            apdu.sendBytesLong( nameBytes , (short) 0 , (short)nameBytes.length );
                        apdu.sendBytesLong( emailBytes , (short) 0 , (short)emailBytes.length );
                        apdu.sendBytesLong( telBytes , (short) 0 , (short)telBytes.length );
            break;
            case WRITE:
            short bytesRead = apdu.setIncomingAndReceive();
              short echoOffset = (short)0;
              while ( bytesRead > 0 )
              {
                Util.arrayCopyNonAtomic(buffer , ISO7816.OFFSET_CDATA , echoBytes ,
echoOffset , bytesRead);
                echoOffset += bytesRead;
                bytesRead = apdu.receiveBytes(ISO7816.OFFSET_CDATA);
            }
            break;
            default :
              ISOException.throwIt(SW_INSTRUCTION_NOT_SUPPORTED);
          }
        break;
        }
    } // 'Applet.process'

} // class 'Frame'
```

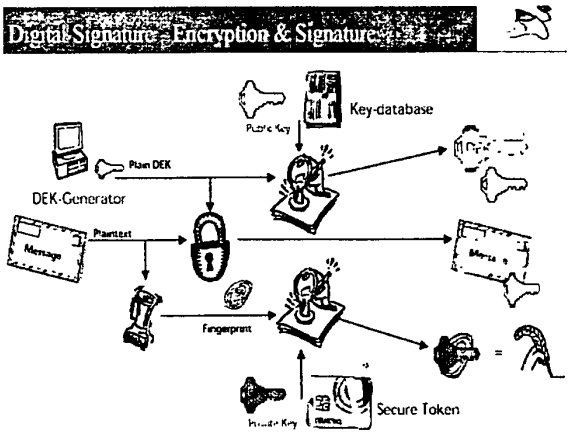
範例二 資料加密

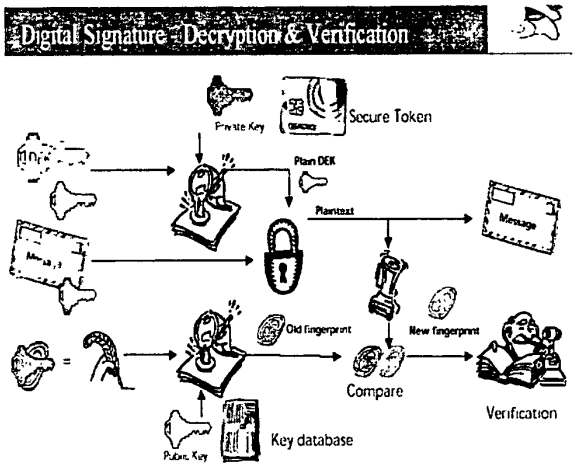
目的：

設計在卡片資料被讀出時具有資料加密功能的 applet。

說明：

本範例將能瞭解 applet 在處理資料加密功能時，所利用到 javacard.security 及 javacardx.crypto 相關物件之方法、函數的設計方式，下圖為一般程式設計時所用到的加解密流程。






```
/*
  EXEC2 Program From G&D Sm@rtcafe Sample
 */
package com.gieseckedevrient.applets.des;
import javacard.framework.*;
import javacard.security.*;
import javacardx.crypto.*;
public class Secure extends Applet
{
    // ---------------------------------------------------------------------
    // static data for e_square
    // ---------------------------------------------------------------------
        // CLA and INS bytes
        private static final byte    ISO_CLA      =   (byte) 0x05;
        private static final byte    ENCR_VERIFY  =   (byte) 0xC0;
        private static final byte    READ_SIGNED  =   (byte) 0xB6;
        private static final byte    READ_ENCRPT  =   (byte) 0xB7;
        // byte arrays for data
```

```
    private static byte[]       m_sba_data;
    private static byte[]       m_sba_pin;
    private static byte[]       m_sba_encr_key;
    private static boolean      m_sz_verified;
    // References for Key ' signature and cipher objects
    private static Signature m_so_signature;
    private static Cipher       m_so_cipher;
    private static DESKey       m_so_key;
// -- END - static elements ----------------------------------------------
// ------------------------------------------------------------------------
// Install method
// ------------------------------------------------------------------------
public static void install(byte[] buffer ' short offset ' byte length)
{
    new Secure();
}
// -- END - Public static void install( ... ) ----------------------------
// ------------------------------------------------------------------------
// Default constructor
// ------------------------------------------------------------------------
public Secure()
{
    // initialize ByteArrays
    // data
    m_sba_data = new byte[ 11 ];
        m_sba_data[  0 ] = (byte)'H';
        m_sba_data[  1 ] = (byte)'e';
        m_sba_data[  2 ] = (byte)'l';
        m_sba_data[  3 ] = (byte)'l';
        m_sba_data[  4 ] = (byte)'o';
        m_sba_data[  5 ] = (byte)' ';
        m_sba_data[  6 ] = (byte)'W';
        m_sba_data[  7 ] = (byte)'o';
        m_sba_data[  8 ] = (byte)'r';
        m_sba_data[  9 ] = (byte)'l';
        m_sba_data[ 10 ] = (byte)'d';
    // PIN
    m_sba_pin = new byte[ 4 ];
```

```
            m_sba_pin[ 0 ] = (byte)'1';
            m_sba_pin[ 1 ] = (byte)'2';
            m_sba_pin[ 2 ] = (byte)'3';
            m_sba_pin[ 3 ] = (byte)'4';
        // key for encryption
        m_sba_encr_key = new byte[ 8 ];
            m_sba_encr_key[ 0 ] = (byte) 0x24;
            m_sba_encr_key[ 1 ] = (byte) 0x52;
            m_sba_encr_key[ 2 ] = (byte) 0xA0;
            m_sba_encr_key[ 3 ] = (byte) 0x6F;
            m_sba_encr_key[ 4 ] = (byte) 0x77;
            m_sba_encr_key[ 5 ] = (byte) 0x6F;
            m_sba_encr_key[ 6 ] = (byte) 0xF0;
            m_sba_encr_key[ 7 ] = (byte) 0x52;
        m_sz_verified = false;
            // provide cipher objects (keys , signatures , ...)
            m_so_key        =
(DESKey)KeyBuilder.buildKey( KeyBuilder.TYPE_DES_TRANSIENT_DESELECT ,
KeyBuilder.LENGTH_DES , false );
            m_so_signature = Signature.getInstance( Signature.ALG_DES_MAC8_ISO9797_M1 ,
false);
            m_so_cipher     = Cipher.getInstance( Cipher.ALG_DES_CBC_ISO9797_M1 , false );
        register();
    }
    // -- END - public MyFirst() ------------------------------------------
    // ------------------------------------------------------------------------
    // Process method
    // ------------------------------------------------------------------------
    public void process( APDU o_apdu ) throws ISOException
    {
        if( selectingApplet() )
        {
            // no verfivation happend
            m_sz_verified   = false;
            // initialize DES key with bytearray
            m_so_key.setKey( m_sba_encr_key , (short)0 );
            return;
        }
```

```
short s_leng = (short)0x0000;
byte[] ba_apdu_buffer = o_apdu.getBuffer();
// Examination of the buffer.
// Switch to class byte
switch( ba_apdu_buffer[ ISO7816.OFFSET_CLA ] )
{
    case ISO_CLA:
        // switch to instruction
        switch( ba_apdu_buffer[ ISO7816.OFFSET_INS ] )
        {
            // Verify with encrypted key
            case ENCR_VERIFY:
                s_leng = receive( o_apdu );
                // initialize cipher object with 'our' key for decryption
                m_so_cipher.init( m_so_key , Cipher.MODE_DECRYPT );
                // decrypt data from apdu buffer
                m_so_cipher.doFinal( ba_apdu_buffer ,
                (short)(ISO7816.OFFSET_CDATA & 0x00FF) , s_leng , ba_apdu_buffer ,
                (short)0 );
                // compare decrypted data with PIN
            s_leng = Util.arrayCompare( ba_apdu_buffer , (short)0 , m_sba_pin , (short)0 ,
short)4 );
                if( s_leng == (byte)0x00 )
                    m_sz_verified = true;
                else
ISOException.throwIt( ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );
                break;
            // read plain data and signature
            case READ_SIGNED:
                if( !m_sz_verified )
ISOException.throwIt( ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );
                // Sign 'Hello World' ...
                m_so_signature.init( m_so_key , Signature.MODE_SIGN );
        s_leng = m_so_signature.sign( m_sba_data , (short)0 , (short)11 , ba_apdu_buffer ,
(short)11 );
        Util.arrayCopy( m_sba_data , (short)0 , ba_apdu_buffer , (short)0 , (short)11 );
                // ... and send
                o_apdu.setOutgoing();
```

```
                    o_apdu.setOutgoingLength( (short)(s_leng + (short)11) );
                o_apdu.sendBytesLong( ba_apdu_buffer , (short)0 , (short)(s_leng + (short)11) );
                    break;
                // get encrypted data
                case READ_ENCRPT:
                    if( !m_sz_verified )
ISOException.throwIt( ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );
        // encrypt 'Hello World' ...
        m_so_cipher.init( m_so_key , Cipher.MODE_ENCRYPT );
    s_leng = m_so_cipher.doFinal( m_sba_data , (short)0 , (short)11 , ba_apdu_buffer , (short)0 );
                    // ... and send
                    o_apdu.setOutgoing();
                    o_apdu.setOutgoingLength( (short)s_leng );
                    o_apdu.sendBytesLong( ba_apdu_buffer , (short)0 , (short)s_leng );
                    break;
                default :
                    // Exception : 0x6D00
                    ISOException.throwIt( ISO7816.SW_INS_NOT_SUPPORTED );
                }
                break;
            default:
                // Exception : 0x6E00
                ISOException.throwIt( ISO7816.SW_CLA_NOT_SUPPORTED );
        }
    } // -- END - public void process( ... ) throws ISOException ----------

    // ---------------------------------------------------------------------
    // receive APDU
    // ---------------------------------------------------------------------
    public short receive( APDU o_apdu ) throws ISOException
    {
        byte[] ba_apdu_buffer = o_apdu.getBuffer();
        // Lc tells the incoming apdu command length.
        short s_readCount = (short)( ba_apdu_buffer[ ISO7816.OFFSET_LC ] & 0x00FF );
        if( s_readCount != o_apdu.setIncomingAndReceive() )
            // Exception : 0x6700
            ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
        return s_readCount;
```

```
    } // -- END - public short receive( ... ) throws ISOException ----------
} // -- END - public class MyFirst extends Applet------------------------------
```

範例三 State Machine 安全性設計

目的：

設計在卡片中有幾種狀態(State) 功能的 applet，依不同 State 模式有不同存取權限及資料加密功能，此程式目的在提供具有安全性功能的卡片設計方法，不同的人依其權限，劃分不同的讀寫功能。

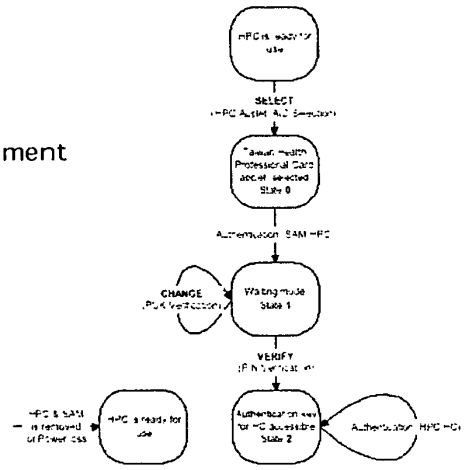
說明：

下圖為 State Machine 流程說明，本範例將能瞭解 applet 在設計一般 State Machine 時，所利用到相關物件之方法、函數。

Definition of the security structure

- Authentication mechanism
- Access conditions for each data element
- Encryption
- Secure messaging
- Definition of states and transitions

⇨ State machine and type of card




```
/*
* Copyright 2002/1/23     EXEC 3 Program Design by Nhib Nembers of Taiwan
*/
package com.gieseckedevrient.applets.myfirst;

import javacard.framework.*;
import javacard.security.*;
import javacardx.crypto.*;

public class MyFirst extends Applet
{
    // ----------------------------------------------------------------------
    // static data for e_square
    // ----------------------------------------------------------------------
        final static byte      ISO_CLA     = (byte) 0x05;
        final static byte      SELECT      = (byte) 0xA4;
```

```
    final static byte      VERIFY      = (byte) 0x20;
    final static byte      READ        = (byte) 0xB6;
    final static byte      MODIFY      = (byte) 0xD6;
    final static byte      GETPIN      = (byte) 0xB7;
    private static byte[]      m_sba_encr_key;
    // References for Key ,  signature and cipher objects
    private static Cipher      m_so_cipher;
    private static DESKey      m_so_key;
    static byte[]   name;
    static byte[]   address;
    static byte[]   content;
    static byte[]   turvover;
    static byte[]   m_sba_aPin;        // Pin for modifying data will be : "1111"
    static byte[]   m_sba_uPin;        // Pin   for reading data   will be : "0000"
    static boolean m_sz_aVerified;
    static boolean m_sz_uVerified;
// -- END - static elements ----------------------------------------------
// ------------------------------------------------------------------------
// Install method
// ------------------------------------------------------------------------
public static void install(byte[] buffer ,  short offset ,  byte length)
{
    new MyFirst();
}
// -- END - Public static void install( ... ) ----------------------------
// ------------------------------------------------------------------------
// Default constructor
// ------------------------------------------------------------------------
public MyFirst()
{
//      short i;
    name=new byte[10];
    name[0]=(byte)'J';
    name[1]=(byte)'a';
    name[2]=(byte)'c';
    name[3]=(byte)'k';
    name[4]=(byte)' ';
    name[5]=(byte)'C';
```

```
            name[6]=(byte)'h';
            name[7]=(byte)'a';
            name[8]=(byte)'n';
            name[9]=(byte)'g';
           // for (i=10;i<10;i++){
          //   name[    (short)i ] = (byte)' ';
//}
            address=new byte[10];
            address[0]=(byte)'B';
            address[1]=(byte)'r';
            address[2]=(byte)'e';
            address[3]=(byte)'t';
            address[4]=(byte)'o';
            address[5]=(byte)'n';
            address[6]=(byte)'i';
            address[7]=(byte)'s';
            address[8]=(byte)'c';
            address[9]=(byte)'h';
            content=new byte[10];
            content[0]=(byte)'0';
            content[1]=(byte)'8';
            content[2]=(byte)'9';
            content[3]=(byte)'5';
            content[4]=(byte)'5';
            content[5]=(byte)'1';
            content[6]=(byte)'0';
            content[7]=(byte)'4';
            content[8]=(byte)'1';
            content[9]=(byte)'1';
            turvover=new byte[6];
            turvover[0]=(byte)'1';
            turvover[1]=(byte)'0';
            turvover[2]=(byte)'0';
            turvover[3]=(byte)'0';
            turvover[4]=(byte)'0';
            turvover[5]=(byte)'0';
            m_sba_aPin = new byte[ 4 ];
            m_sba_aPin[ 0 ] = (byte) '1';
```

```
            m_sba_aPin[ 1 ] = (byte) '1';
            m_sba_aPin[ 2 ] = (byte) '1';
            m_sba_aPin[ 3 ] = (byte) '1';
            m_sba_uPin = new byte[ 4 ];
            m_sba_uPin[ 0 ] = (byte) '0';
            m_sba_uPin[ 1 ] = (byte) '0';
            m_sba_uPin[ 2 ] = (byte) '0';
             m_sba_uPin[ 3 ] = (byte) '0';
            m_sz_aVerified = false;
            m_sz_uVerified = false;
///-------------------------------------------------------------
    // key for encryption
            m_sba_encr_key = new byte[ 8 ];
                m_sba_encr_key[ 0 ] = (byte) 0x24;
                m_sba_encr_key[ 1 ] = (byte) 0x52;
                m_sba_encr_key[ 2 ] = (byte) 0xA0;
                m_sba_encr_key[ 3 ] = (byte) 0x6F;
                m_sba_encr_key[ 4 ] = (byte) 0x77;
                m_sba_encr_key[ 5 ] = (byte) 0x6F;
                m_sba_encr_key[ 6 ] = (byte) 0xF0;
                m_sba_encr_key[ 7 ] = (byte) 0x52;

                // provide cipher objects (keys , signatures , ...)
                m_so_key          =
(DESKey)KeyBuilder.buildKey( KeyBuilder.TYPE_DES_TRANSIENT_DESELECT ,
KeyBuilder.LENGTH_DES , false );
                m_so_cipher       = Cipher.getInstance( Cipher.ALG_DES_CBC_ISO9797_M1 ,
false );
            register();
    }
    // -- END - public MyFirst() ----------------------------------------
    // ------------------------------------------------------------------------
    // Process method
    // ------------------------------------------------------------------------
    public void process(APDU o_apdu) throws ISOException
    {
        if( selectingApplet() )
        {
```

```
            m_sz_aVerified = false;
            m_sz_uVerified = false;

            // initialize DES key with bytearray
            m_so_key.setKey( m_sba_encr_key , (short)0 );
            return;
        }
        short s_dataoffset = -10;
        short s_datalength = -20;
        short s_databytes  = -30;
        byte  b_result1    = (byte) 0x11;
        byte  b_result2    = (byte) 0x11;
        short s_leng = (short)0x0000;
        byte[] ba_buffer = o_apdu.getBuffer();
        //Examination of the buffer.
        // Switch to class byte
        switch( ba_buffer[ ISO7816.OFFSET_CLA ] )
        {
        case ISO_CLA:
        switch( ba_buffer[ ISO7816.OFFSET_INS ] )
        {
            //PIN Verification as defined in ISO 7816-4.
            case VERIFY: // ----------------------------------------------------
                s_databytes  = receive( o_apdu );
                s_dataoffset = Util.getShort( ba_buffer , ISO7816.OFFSET_P1 );
             b_result1 = Util.arrayCompare( ba_buffer , (short) (ISO7816.OFFSET_CDATA &
0x00FF) , m_sba_aPin , (short)0 , s_databytes);
                b_result2 = Util.arrayCompare( ba_buffer , (short) (ISO7816.OFFSET_CDATA &
0x00FF) , m_sba_uPin , (short)0 , s_databytes);
                if( b_result2 == (byte)0x00 )
                {  m_sz_uVerified = true;
                    m_sz_aVerified = false;  }
                else if( b_result1 == (byte)0x00 )
                { m_sz_uVerified = false;
                  m_sz_aVerified = true;  }
                else
               ISOException.throwIt( ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED );
                break;
```

```
case READ: // ----nopin send basic data     ------------------------------------
        o_apdu.setOutgoing();
        o_apdu.setOutgoingLength( (short) 30 );
        o_apdu.sendBytesLong( name , (short) 0 , (short)10 );
        o_apdu.sendBytesLong( address , (short) 0 , (short)10 );
        o_apdu.sendBytesLong( content , (short) 0 , (short)10 );
//       o_apdu.sendBytesLong( turvover , (short) 0 , (short)6 );
        break;
    case GETPIN: // -- getpin 0000 and send encryp turvover data ----------
        if( !m_sz_uVerified && !m_sz_aVerified )
ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
            // encrypt 'turvover' ...
            m_so_cipher.init( m_so_key , Cipher.MODE_ENCRYPT );
        s_leng = m_so_cipher.doFinal( turvover , (short)0 , (short)6 , ba_buffer ,
short)0 );
            // ... and send
            o_apdu.setOutgoing();
            o_apdu.setOutgoingLength( (short)s_leng );
            o_apdu.sendBytesLong( ba_buffer , (short)0 , (short)s_leng );
        break;
    case MODIFY: // -- getpin 1111 and modify all data ----------------------------------
        if( !m_sz_aVerified )
        ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
        b_result1     = 0x00;
        s_databytes   = receive( o_apdu );
        s_dataoffset = ISO7816.OFFSET_CDATA;
        // set name
    while( ba_buffer[ s_dataoffset ] != (byte)0xFF )
        {   name[b_result1] = ba_buffer[ s_dataoffset ];
            b_result1++;
            s_dataoffset++;   } // and fill with ASCII-Spaces
        while( b_result1 < (short) 10 )
        {    name[ b_result1 ] = (byte)0x20;
            b_result1++; }
        b_result1     = 0x00;
        s_dataoffset++;
        // set address
        while( ba_buffer[ s_dataoffset ] != (byte)0xFF )
```

```
            {
                address[b_result1] = ba_buffer[ s_dataoffset ];
                b_result1++;
                s_dataoffset++;
            } // and fill with ASCII-Spaces
            while( b_result1 < (short) 10 )
            {
                address[ b_result1 ] = (byte)0x20;
                b_result1++;
            }
            b_result1       = 0x00;
            s_dataoffset++;
            // set content
            while( ba_buffer[ s_dataoffset ] != (byte)0xFF )
            {
                content[b_result1] = ba_buffer[ s_dataoffset ];
                b_result1++;
                s_dataoffset++;
            }
            b_result1       = 0x00;
            s_dataoffset++;
            // set turvover
            while( ba_buffer[ s_dataoffset ] != (byte)0xFF )
            { turvover[b_result1] = ba_buffer[ s_dataoffset ];
                b_result1++;
                s_dataoffset++;   }
            break;
        default : // ----------------------------------------------------
            ISOException.throwIt( ISO7816.SW_INS_NOT_SUPPORTED );
    }
        break;
        default:
            // Exception : 0x6E00
            ISOException.throwIt( ISO7816.SW_CLA_NOT_SUPPORTED );
    }
} // -- END - public void process( ... ) throws ISOException ----------
// ------------------------------------------------------------------------
// receive APDU
```

```
    // ---------------------------------------------------------------------
    public short receive(APDU o_apdu) throws ISOException
    {
        byte[] ba_buffer = o_apdu.getBuffer();
        //Lc tells the incoming apdu command length.
        short s_readCount = (short) (ba_buffer[ ISO7816.OFFSET_LC ] & 0x00FF);
        if( s_readCount != o_apdu.setIncomingAndReceive() )
            ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
        return s_readCount;
    } // -- END - public short receive( ... ) throws ISOException ------------
} // -- END - public class MyFirst extends Applet------------------------------
```