**File Structure**

The files on a smart card are organized in a tree structure. The topmost file is the Master File (MF). The MF has one or more Application Definition Files (ADF). Inside of an ADF are Applicaton Elementary Files (AEF) that contain data.  
You can quickly select an ADF with the Application Identifier (AID). Within an ADF you can select AEFs with the Short File Identifier (SFI).

**APDU - Application Protocol Data Unit**

After the reset, the communication between terminal and card works with APDUs.

**Command APDU**

The terminal sends a command APDU to the card. This command has a mandatory header and an optional body.

| **CLA** | **INS** | **P1** | **P2** | **Lc** | **Data** | **Le** |
| --- | --- | --- | --- | --- | --- | --- |
| Header | | | | Trailer | | |

|  |  |
| --- | --- |
| **Field** | **Description** |
| CLA | Class byte 0x00 |
| INS | Instruction byte 0xA4:Select Command 0xB2:Read Record Command |
| P1 | Parameter 1 byte The value and meaning depends on the instruction code (INS). |
| P2 | Parameter 2 byte The value and meaning depends on the instruction code (INS). |
| Lc | Number of data bytes send to the card. With the Smart Card Shell the value of Lc will be calculated automatically. You don't have to specify this parameter. |
| Data | Data byte |
| Le | Number of data bytes expected in the response. If Le is 0x00, at maximum 256 bytes are expected. |

**Response APDU**

The card will execute the command and send a response APDU back to the terminal. The response APDU has an optional body consisting of data and a mandatory trailer with two status bytes "SW1" and "SW2". SW1 and SW2 combined are the status word (SW). If the status word has the value 0x9000 (SW1 = 0x90, SW2=0x00), the command was successfully executed by the card.

| **Data** | **SW 1** | **SW 2** |
| --- | --- | --- |
| Body | Trailer | |

**Case 1 to 4**

There are four different cases of APDU:

Case 1

Command: Header  
Response: Trailer

Case 2

Command: Header + Le  
Response: Data + Trailer

Case 3

Command: Header + Data  
Response: Trailer

Case 4

Command: Header + Data + Le  
Response: Data + Trailer

**Read EMV Script**

To read all data from the card, you can use the script contained in the following section:

try

{

var card = new [Card](http://www.openscdp.org/scsh3/card.html)(\_scsh3.reader);

card.[reset](http://www.openscdp.org/scsh3/card.html#reset)(Card.RESET\_COLD);

var aid = new [ByteString](http://www.openscdp.org/scsh3/bytestring.html)("A0000000041010", HEX); // MC

// var aid = new ByteString("A0000000031010", HEX); // VISA

// var aid = new ByteString("1PAY.SYS.DDF01", ASCII);

var fcp = card.[sendApdu](http://www.openscdp.org/scsh3/card.html#sendApdu)(0x00, 0xA4, 0x04, 0x00, aid, 0x00, [0x9000]);

print("FCP returned in SELECT: ", new ASN1(fcp));

for (var sfi = 1; sfi <= 31; sfi++)

{

for (var rec = 1; rec <= 16; rec++)

{

var tlv = card.sendApdu(0x00, 0xB2, rec, (sfi << 3) | 4, 0x00);

if (card.SW == 0x9000)

{

print("SFI " + sfi.toString(16) + " record #" + rec);

try

{

var asn = new [ASN1](http://www.openscdp.org/scsh3/asn1.html)(tlv);

print(asn);

}

catch(e)

{

print(tlv.toString(HEX));

}

}

}

}

}

catch(e)

{

print("Exception reading from Credit Card Application: " + e.toString());

}

Line 06  
This script works either with Mastercard or VISA because both cards are using different AIDs. You'll need to activate the AID that matches your card.

Line 13  
To read the card's data we have to select the right ADF first. We'll do this with a SELECT command encoded as case 4 command APDU and transmitted using card.sendApdu():  
The SELECT command supports the following options :

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| 0x00 | Class byte |
| 0xA4 | Instruction byte SELECT Command |
| 0x04 | Parameter 1 byte Select a directory by name. |
| 0x00 | Parameter 2 byte In general P2 is set to 0x00 when you write the complete AID Name. |
| aid | Data bytes contain the directory name (AID) |
| 0x00 | Le byte Set to 0x00 to retrieve the complete response of up to 256 bytes. |
| [0x9000] | Array of acceptable SW1/SW2 return codes |

The card.sendAPDU() method supports a further parameter, which is used to define a list of expected SW codes. [0x9000] defines a list with a single 0x9000 entry.

Line 17  
Now we create a loop that starts with the first AEF (SFI=1) and ends with SFI 31.

Line 19  
In a second loop the record number of every AEF will be iterated from 1 to 16.

Line 21  
With the READ RECORD command issued using card.sendApdu() we can obtain the data.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| 0x00 | Class byte |
| 0xB2 | Instruction byte READ RECORD Command |
| rec | Parameter 1 byte. It contains the record number. |
| (sfi << 3) | 4 | Parameter P2 byte. The first 5 bits contain the SFI and the last three bits are set to 1,0,0, which means that P1 is a record number. To calculate the P2 parameter, the SFI will be shift three bits to the left and combinated with a '4' e.g. 00000001 00001000 <<3 00000100 | 4 00001100 = 0x0C |
| 0x00 | Le byte. It is '00' so up to 256 bytes are expected. |

Line 22  
If the statusword has the value 0x9000 the command was successful and the tlv object will be printed.