TSG-Terminals Working Group 3 (USIM) meeting #7
5-7 July, 1999

TSGT3#7(99)187
Page 1

Source: Schlumberger

Title: SCHLUMBERGER PROPOSAL OF 3RD GENERATION PHONE BOOK FEATURE

Tables of contents

SCOPE

This document is intended to provide a solution that can be deliverable within the existing capabilities of the smart card hardware in term of RAM size and microprocessor speed, etc…

The best way to be able to stay compatible with 2G is to create another type of linear fixed file where the number of record is coded in 16 bits, and use cyclic structure.

Note: Cyclic structure:

A structure is set of records in different files. A cyclic structure is the same set of records linked through the pointer's chain where the last one is linked to the first to form a token.

Requirements

Definition of Phone Book

The Phone Book feature is based on the ADN functionality as defined in GSM 11.11 [7]. Additional features are identified in the following sub-clauses. A Phone Book entry consists of a record in an ADN file and, optionally, additional records which are placed in different EFs. In the latter case, a mechanism shall be defined to link all records in the same Phone Book entry. The ME shall support these features while their support by the USIM is optional.

Support of two name fields per entry

The support of two name fields per entry shall be specified to allow, for example, for two different representations of the same name, for example, in Japanese and English.

Support of multiple phone numbers per entry

The support of multiple phone numbers per entry shall be specified, for example office, home, fax, mobile or pager. In addition to that, information for identifying those attributes is needed.

Support of email address

The support of email addresses linked to Phone Book entries shall be specified. In addition to that, information for identifying these addresses is needed.

Support of user definable groupings

The specification shall support the grouping of Phone Book entries into groups defined by the user, for example, business and private.

Support of hidden entries

The specification shall support means of marking Phone Book entries as "hidden".

Number of entries

The specification shall support storage of at least 500 entries.

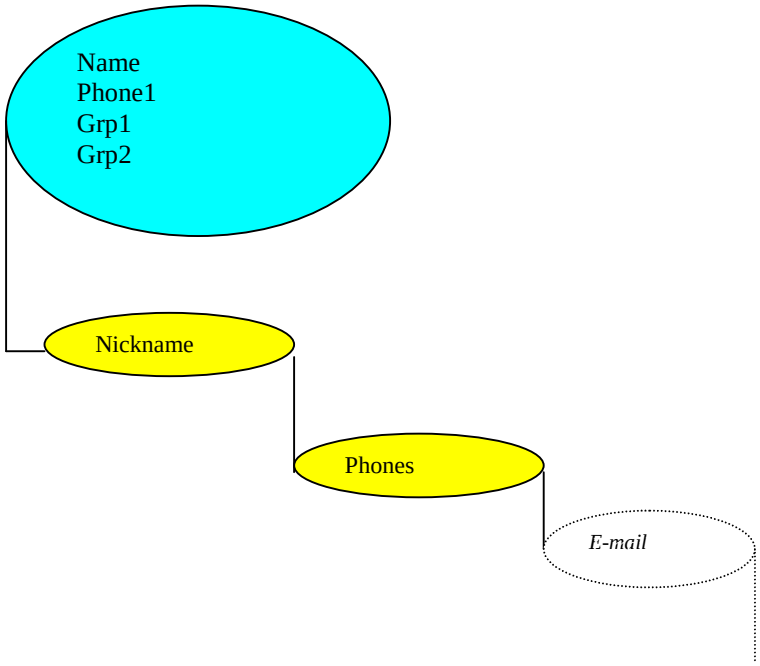
Mode of alerting

[FFS]

Solution

To be compliant with the 21.111 v3.00 we need to include the phonebook functionality as an application in the USIM where the limitation of the structure “definition” of the files as are described in 7816-3(4) does not constraint the functionality.

The solution will be transparent for the external world (ME). Special APDUs will be adapted to handle the application, for each entry in the phone book the structure will be linked to one nickname or none (and/or) to “0-4 “extra phone numbers, (and/or) to one E-mail address or none.



Phonebook Entry →{		*Name*	*14 Bytes;*	*X1*
		Phone1	*14 Bytes;*	*X2*
		Grp1, Grp2	*1 Bytes;*	*X3*
		Pointer P	*2 Bytes;*	*X4*
	}			
Nickname {		*Nickname*	*5 Bytes;*	*X5*
		Pointer	*2 Bytes;*	*X6*
}				
Phones	**{**	*Ref*	*1Bytes;*	*X7*
		Phone2	*8Bytes;*	*X8*
		Phone3	*8Bytes;*	*X9*
		Phone4	*8Bytes;*	*X10*
		Phone5	*8Bytes;*	*X11*
		Pointer	*2Bytes;*	*X12*
}				
E-mail	**{**			
		Email	*30 Bytes;*	*X13*
		Pointer L	*2Bytes;*	*X14*
}				

Structure of the Files

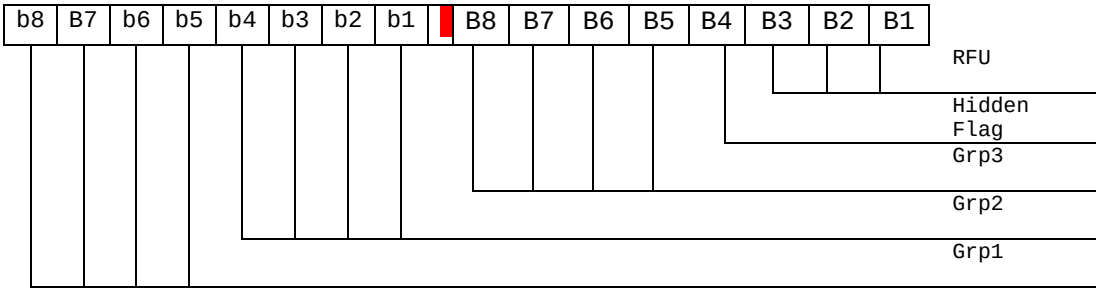
Phonebook Entry

Table 1 - Structure of EF_{EPB}

File Identifier: 'XXXX'	Structure: extended linear fixed	Mandatory
Record length: X+17 bytes	Update activity: low	
Access Conditions: READ CHV1 UPDATE CHV1 INVALIDATE CHV2 REHABILITATE CHV2		

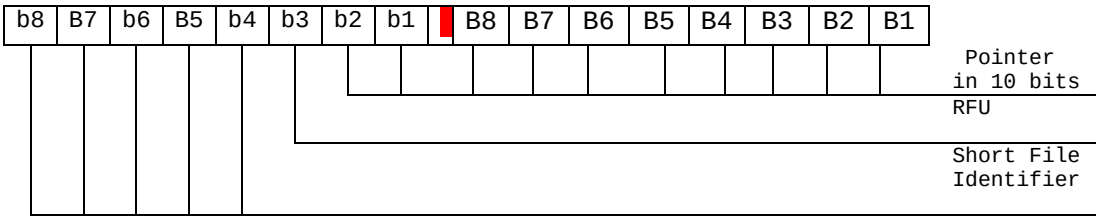
Bytes	Description	M/O	Length
1 to X	Alpha Identifier (*Name*)	O	X bytes
X+1	Length of BCD number/SSC contents	M	1 byte
X+2	TON and NPI	M	1 byte
X+3 to X+12	Dialling Number/SSC String	M	10 bytes
X+13	Capability/Configuration Identifier	M	1 byte
X+15	Grouping	M	2 bytes
X+17	Pointer P	M	2 bytes

Coding of Grouping



Note1 :From B4 to B4: 1 bit: if b3=0 the entry is not hidden, if b3= 1 the entry hidden.

Coding of Pointer P



Note1:

From b8 to b4: 5 bits: to define Short File Identifier where the pointer b2 to B1 is pointing to.
From b2 to B1: 10 bits: to define the value of the record number in the file identified by SFI in b8 to b4.

Note2:

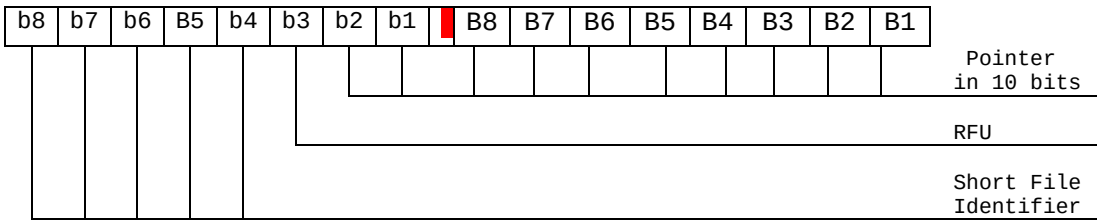
The pointer "*pointer P*" can be connected to Nickname file or Phones file or E-mail file or back to entry of the Phone Book.

Nickname

Table 2 - Structure of $EF_{Nickname}$

File Identifier: 'XXXX'	Structure: extended linear fixed		Optional
Record length: X +2 bytes		Update activity: low	
Access Conditions: READ CHV1 UPDATE CHV1 INVALIDATE CHV2 REHABILITATE CHV2			
Bytes	Description	M/O	Length
1 to X	Alpha Identifier (*Nickname*)	O	X bytes
X+1to X+2	Pointer	M	2 bytes

Coding of Pointer



Note1:

From b8 to b4: 5 bits: to define Short File Identifier where the pointer b2 to B1 is pointing to.
From b3 to b3: 1 bit: RFU.
From b2 to B1: 10 bits: to define the value of the record number in the file identified by SFI in b8 to b4.

Note2:

The pointer "*pointer P*" can be connected to Phones file or E-mail file or back to entry of Phone Book.

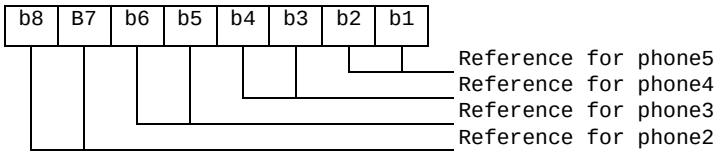
Phones

Table 3 - Structure of EF_{Phones}

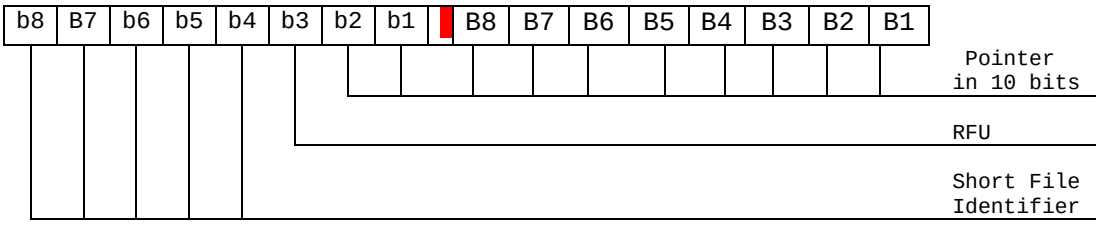
File Identifier: 'XXXX'	Structure: extended linear fixed	Optional
Record length: X bytes=13,23,33,43	Update activity: low	
Access Conditions: READ CHV1 UPDATE CHV1 INVALIDATE CHV2 REHABILITATE CHV2		

Bytes	Description	M/O	Length
1 to 1	Ref	M	1 bytes
2 to 11	Phone2	M	10 bytes
12 to 21	Phone3	M	10 bytes
22 to 31	Phone4	M	10 bytes
32 to 41	Phone5	M	10 bytes
42 to 43	Pointer	M	2 bytes

Coding of Ref



Coding of Pointer



Note1:

From b8 to b4: 5 bits: to define Short File Identifier where the pointer b2 to B1 is pointing to.
From b3 to b3: 1 bit: RFU.
From b2 to B1: 10 bits: to define the value of the record number in the file identified by SFI in b8 to b4.

Note2:

The pointer "*pointer P*" can be connected E-mail file or back to entry of Phone Book.

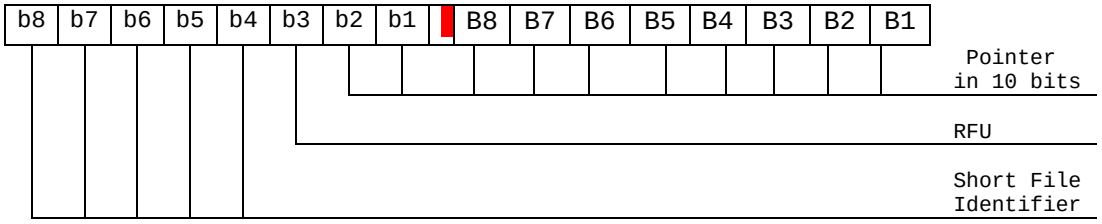
E-mail

Table 4 - Structure of EF_{Email}

File Identifier: 'XXXX'	Structure: extended linear fixed	Optional
Record length: X+2 bytes	Update activity: low	
Access Conditions: READ CHV1 UPDATE CHV1 INVALIDATE CHV2 REHABILITATE CHV2		

Bytes	Description	M/O	Length
1 to X	Alpha Identifier (*E-mail address*)	O	X bytes
X+1 to X+2	Pointer L	M	2 bytes

Coding of Pointer L



Note1:

From b8 to b4: 5 bits: to define Short File Identifier where the pointer b2 to B1 is pointing to.
From b3 to b3: 1 bit: RFU.
From b2 to B1: 10 bits: to define the value of the record number in the file identified by SFI in b8 to b4.

Note2:

The pointer "*pointer P*" can be connected only back to entry of Phone Book.

Reference files:

To be able to support this structure some administrative files need to be created.

Group Reference

This file will contain all the names of the groups, which every phone book entry could belong to.
The content of Group reference file is shown in Table 5.

Table 5 - Group reference File Structure

Identifier: 'XXXX'	Structure: linear fixed	Mandatory
Record Length: 3 bytes	Update activity: low	
Access Conditions: READ CHV1 UPDATE ADM INVALIDATE ADM REHABILITATE ADM		

Bytes	Description	M/O	Length
1 – 3	Group reference	O	3 bytes

Table 6 - Content of Group reference file.

Record number	Group name
01	PER
02	JOB
…	…
0F	XXX

15 records maximum

Reference of the Phone book files

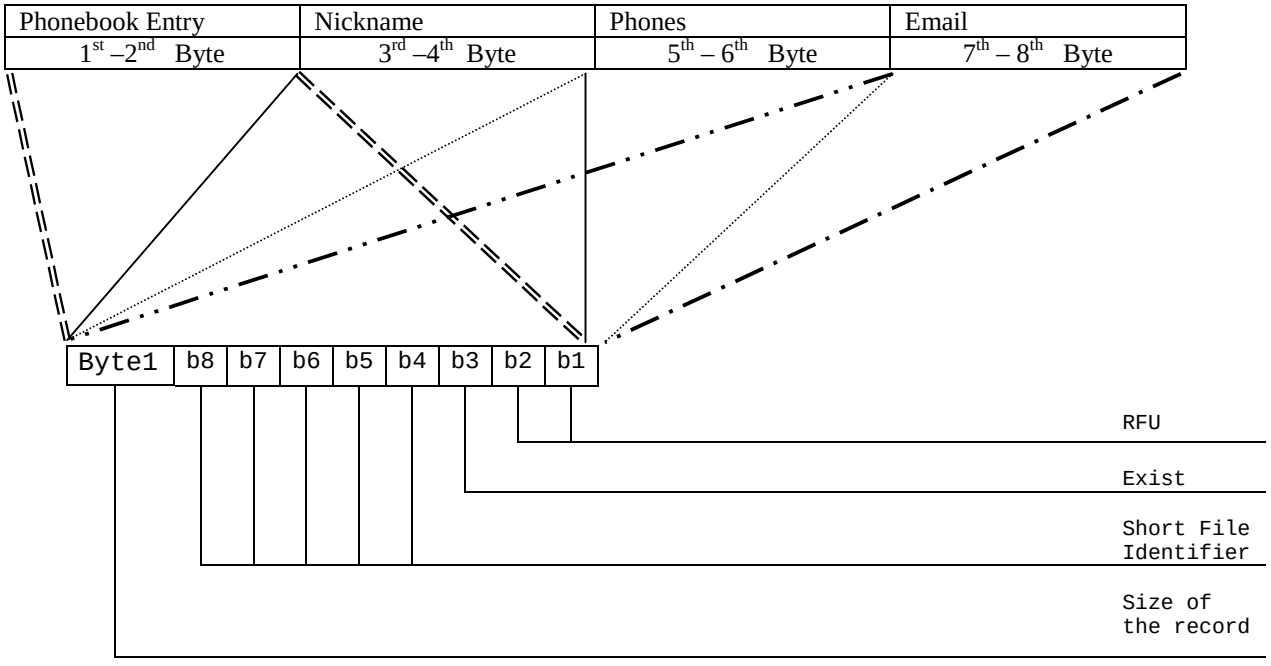
This file will contain all the files' SFI , and their existence in the structure.

Table 7 - Phone Book file Structure

Identifier: 'XXXX'	Structure: Transparent	Mandatory
size: X bytes	Update activity: low	
Access Conditions: READ CHV1 UPDATE ADM INVALIDATE ADM REHABILITATE ADM		

Bytes	Description	M/O	Length
1 – X	Structure	M	X bytes

Coding of Structure:



Note1:

First byte is the size of the record in the corresponding file identified by the following SFI.
From b8 to b4: 5 bits to define Short File Identifier of the corresponding file.
From b3 to b3: b3=0 if the file exists, b3= 1 if it does not exist.
From b2 to b1: 2 bits RFU

Phones Reference

This file will contain all the designation of the extra phone numbers, which will be stored in the phone book, as FX for fax number, HM for home number, etc….

The content of Phones reference file is shown Table 9.

Table 8 - Phones reference File Structure

Identifier: 'XXXX'	Structure: linear fixed	Mandatory
Record Length: 2 bytes	Update activity: low	
Access Conditions: READ CHV1 UPDATE ADM INVALIDATE ADM REHABILITATE ADM		

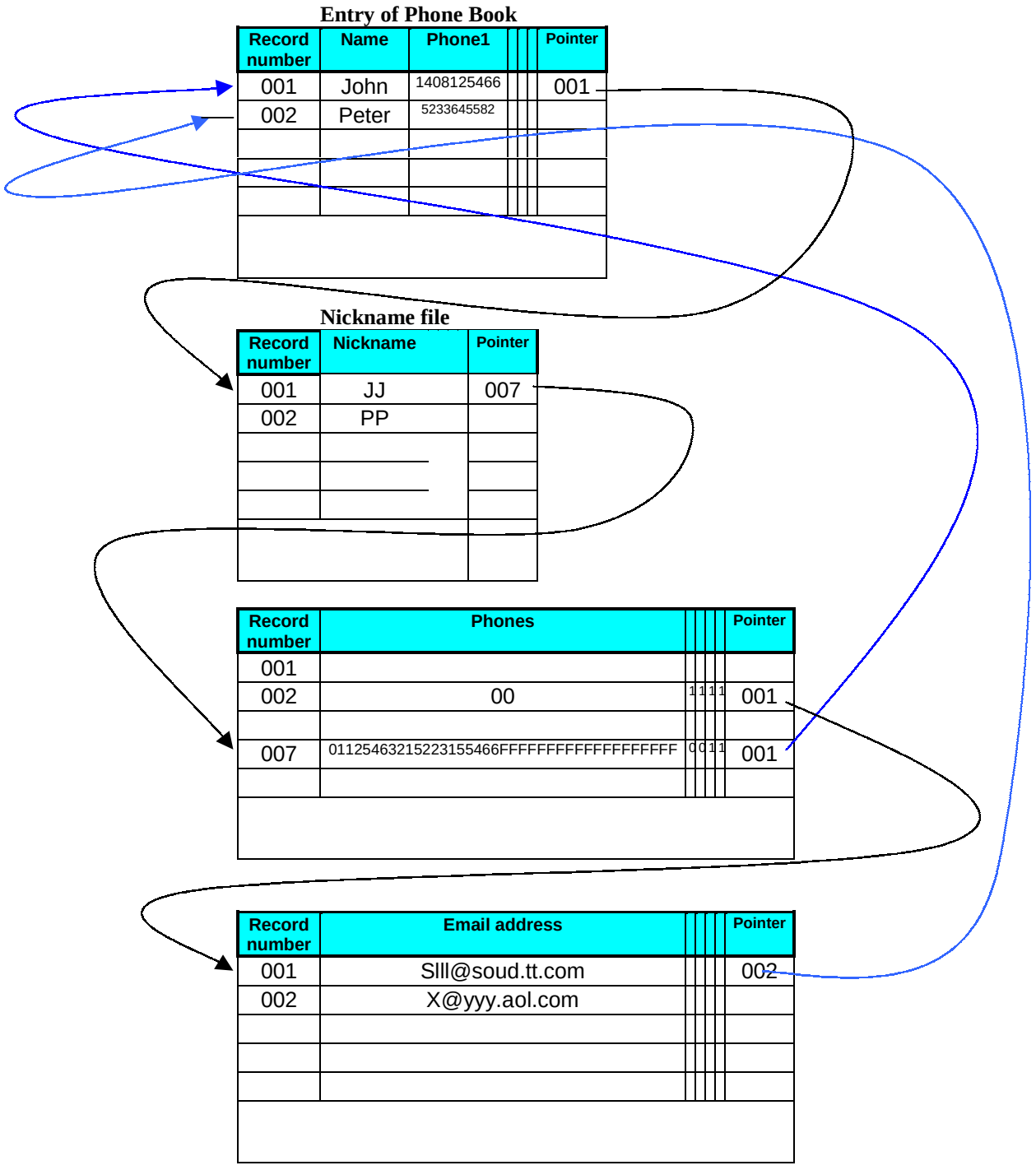
Bytes	Description	M/O	Length
1 – 2	Phone reference	O	2 bytes

Table 9 - Content of Phones reference file.

Record number	Designation
01	FX
02	HM
03	…
04	XX

4 records maximum

Phone book structure



The maximum number of records will be 3FF.

Implementation

Administration commands

Create phonebook

- Create file PHONE BOOK FILE STRUCTURE: it's a reference file.
- Create all the reference files.
- Create all the files needed for the phone book: the EF_{EPB} file is mandatory, all others are optional.
- All the files have to be under the same DF to able to use SFI to access elementary files.

Operation commands

Retrieve a complete structure

Scenario

SELECT PHONE BOOK DIRECTORY

Select the references files and store the content in the phone.

SELECT PHONE BOOK ENTRY

INPUT:

Name of the file

OUTPUT:

SW1=XX
SW2="YY" length of data to retrieve

GET RESPONSE

To get the length of the record and the number of records.

READ RECORD

INPUT:

From record number X
To record number Y
The length of the retrieved data has to be less than 255 *"(Y-X)*length of each record <255"*.

OUTPUT:
Records of the Phonebook entry.

Retrieving other information for the same entry:

Since we have the pointer with the Short File Id and the number of the record of the next data ,we need to send READ RECORD with the right SFI and right record number.

UPDATE RECORD

Command Message

APDU HEADER
CLA = 'AX'. X as defined in 7816-4
INS= 'DC'
P1-P2 See Below

APDU BODY
Interpretation of the Body APDU depends on b3, b2, b1 of P2
L_c :
Case 1 (Simple) : Length of the Data to be overwritten
Case 2 (Extended): 1+ Length of the Data to be overwritten
Data Field:
Case 1 (Simple) : Data to be overwritten
Case 2 (Extended): First byte is the lowest Byte of the Record Number
Subsequent bytes

P1-P2
Interpretation of P1-P2 depends on b3, b2, b1 of P2:

-Case 1 (Simple):
If b3, b2, b1 of P2 =100 then P1 codes the Record Number to be updated (Up to 256 entries). Coding of P2 is the same as 7816-4.
-Case 2 (Extended):
If b3, b2, b1 of P2 =111, then the Record Number is coded over 2 bytes as [P1 || First Data Field byte].

Response Message

As defined in ISO 7816-4

COMMENT For Case 1 (Record Number <255), this command (Structure & Coding) is the same as defined in 7816-4

SEEK (SEARCH RECORD) Command

After a succesful SEEK the record pointer is set to the Record in which the pattern was found.The response Data Field contains the Record Number over 2 bytes.

Command Message

CLA = ‘AX’. X as defined in 7816-4
INS= ‘A2’
P1-P2 See the text below
L_C : Length of the subsequent data field coded over one byte
Data Field: String Data Pattern to be found
L_E = Empty or ‘02’

P1=’00’
P2 specifies type and mode
‘x0’ = Search from the Beginning of the Phonebook Forward
‘x1’ = Search from the End of the Phonebook Backwards
‘x2’ = Search from the Next Current Pointer Position Forward
‘x3’ = Search from the Previous Current Pointer Position Backwards
Where x=’0’ specifies type 1 SEEK and x=’1’ type 2 SEEK.

Response Message

Data Field : Empty or Record Number where the match is found coded over 2 bytes
SW1-SW2 Status Bytes
The following specific warning condition may occur
SW1=’62’ with SW2= ‘82’
SW2= ‘82’ Means end of file reached before finding matching string.The Response Data Field shall be empty.

READ RECORD

Command Message

APDU HEADER
CLA = ‘8X’. X as defined in 7816-4
INS= ‘D2’
P1-P2 See Text below

APDU BODY
Interpretation of the Body APDU depends on b3 of P2
L_c = Empty or ‘01’
Data Field : -Empty (Case 1)
-1 Byte Lowest Byte of the Record Number to be Read (Case 2)
L_E = Length of the Data to be Read over 1 byte.

P1-P2
Interpretation of P1-P2 depends on b3 of P2:

-Case 1 (Simple): If b3 of P2 =1 then P1 codes the Record Number (up to 256 entries. Coding of P2 is the same as 7816-4. No Data Field is present.

-Case 2 (Extended): If b3 of P2 =0, then the Record Number is coded over 2 bytes as [P1 || Data Field byte].

P1= ‘00’ refers to the current Record.

Response Message

As defined in ISO 7816-4

COMMENT For Case 1 (Record Number <255), this command (Structure & Coding) is the same as defined in 7816-4

Compatibility 2G/3G

The ICC will make the difference between the two environments 2G and 3G because of the way the application “environment” is selected. The ICC will have a filter in 2G environment to catch all the commands related to ADN and it will map them to have the information from phonebook file.

Annex

Extended-linear fixed EF

An EF with linear fixed structure consists of a sequence of records all having the same (fixed) length. The first record is record number 1. The length of a record as well as the value its length multiplied by the number of records are indicated in the header of the EF.

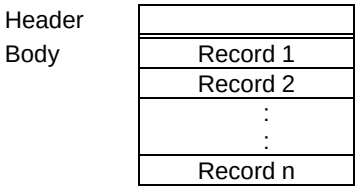


Figure 1 - Structure of a linear fixed file

There are several methods to access records within an EF of this type:

- absolutely using the record number;
- when the record pointer is not set it shall be possible to perform an action on the first or the last record by using the NEXT or PREVIOUS mode;
- when the record pointer is set it shall be possible to perform an action on this record, the next record (unless the record pointer is set to the last record) or the previous record (unless the record pointer is set to the first record);
- by identifying a record using pattern seek starting:
  - forwards from the beginning of the file;
  - forwards from the record following the one at which the record pointer is set (unless the record pointer is set to the last record);
  - backwards from the end of the file;
  - backwards from the record preceding the one at which the record pointer is set (unless the record pointer is set to the first record).

If an action following selection of a record is aborted, then the record pointer shall remain set at the record at which it was set prior to the action.

NOTE 1: It is possible to have more than 255 records in a file of this type because the number of record will be coded in 16 bits, and each record cannot be greater than 255 bytes.

READ RECORD

This function reads one complete record in the current Extended-linear fixed EF. The record to be read is described by the modes below. This function shall only be performed if the READ access condition for this EF is satisfied. The record pointer shall not be changed by an unsuccessful READ RECORD function.
Four modes are defined:

CURRENT: The current record is read. The record pointer is not affected.

ABSOLUTE: The record given by the record number is read. The record pointer is not affected.

NEXT: The record pointer is incremented before the READ RECORD function is performed and the pointed record is read. If the record pointer has not been previously set within the selected EF, then READ RECORD (next) shall read the first record and set the record pointer to this record.

If the record pointer addresses the last record in a Extended-linear fixed EF, READ RECORD (next) shall not cause the record pointer to be changed, and no data shall be read.

PREVIOUS: The record pointer is decremented before the READ RECORD function is performed and the pointed record is read. If the record pointer has not been previously set within the selected EF, then READ RECORD (previous) shall read the last record and set the record pointer to this record.

If the record pointer addresses the first record in a Extended-linear fixed EF, READ RECORD (previous) shall not cause the record pointer to be changed, and no data shall be read.

Input:
- mode, record number (absolute mode only) and the length of the record.

Output:
- the record.

UPDATE RECORD

This function updates one complete record in the current Extended-linear fixed EF. This function shall only be performed if the UPDATE access condition for this EF is satisfied. The UPDATE can be considered as a replacement of the relevant record data of the EF by the record data given in the command. The record pointer shall not be changed by an unsuccessful UPDATE RECORD function.
The record to be updated is described by the modes below. two modes are defined.

CURRENT: The current record is updated. The record pointer is not affected.

ABSOLUTE: The record given by the record number is updated. The record pointer is not affected.

Input:
- mode, record number (absolute mode only) and the length of the record;
- the data used for updating the record.

Output:
- none.

SEEK

This function searches through the current linear fixed EF to find a record starting with the given pattern. This function shall only be performed if the READ access condition for this EF is satisfied. Two types of SEEK are defined:

Type 1 The record pointer is set to the record containing the pattern, no output is available.

Type 2 The record pointer is set to the record containing the pattern, the output is the record number.

The SIM shall be able to accept any pattern length from 1 to 16 bytes inclusive. The length of the pattern shall not exceed the record length.
Four modes are defined:

- from the beginning forwards;
- from the end backwards;
- from the next location forwards;
- from the previous location backwards.

If the record pointer has not been previously set (its status is undefined) within the selected Extended-linear fixed EF, then the search begins:

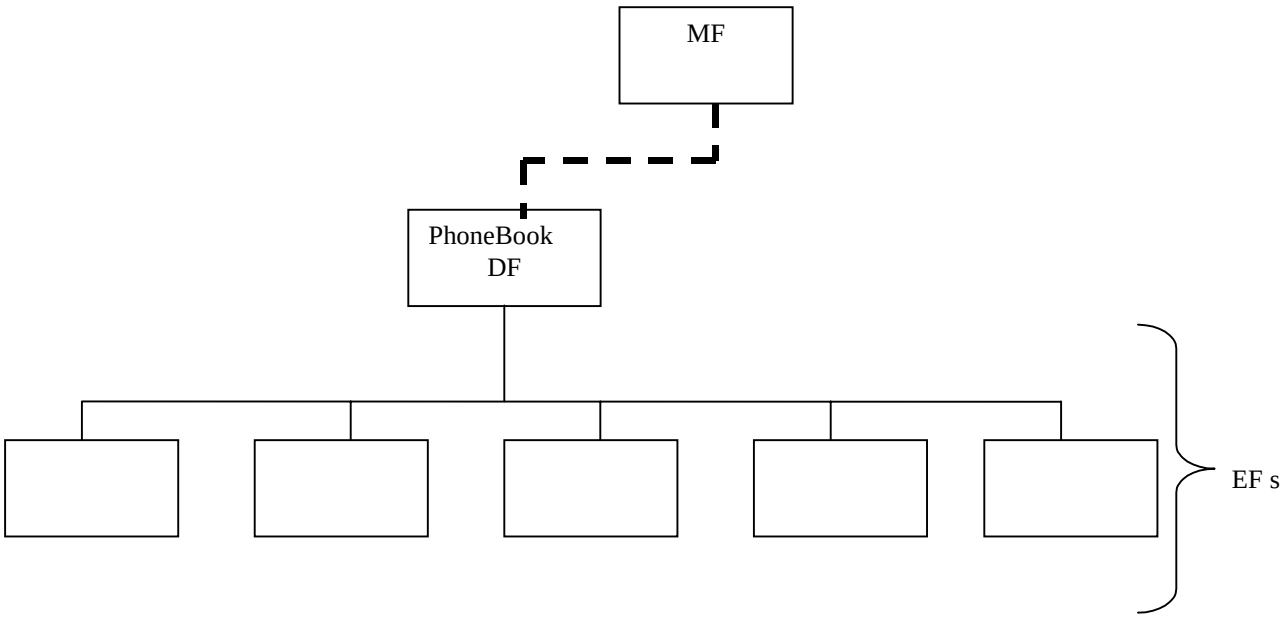
- with the first record in the case of SEEK from the next location forwards; or
- with the last record in the case of SEEK from the previous location backwards.

After a successful SEEK, the record pointer is set to the record in which the pattern was found. The record pointer shall not be changed by an unsuccessful SEEK function.

Input:
- type and mode;
- pattern;
- length of the pattern.

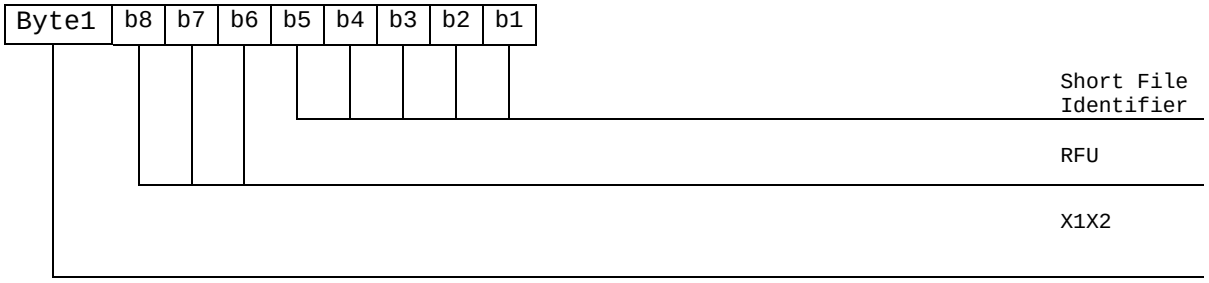
Output:
- type 1: none;
- type 2: status/record number

CODING OF SFI



File Id of Phonebook DF is $X_1X_2X_3X_4$

All the EFs under $X_1X_2X_3X_4$ will have an Id starting with X_1X_2 and second byte coded on 5 bits



Note1:

First byte is the first byte of the parent Id
From b8 to b6: 3 bits RFU.
From b5 to b1: 5 bits to define Short File Identifier of the corresponding file.

Scenarios

3G phones
Add a new entry
Assumptions

- Phonebook DF Id : $X_1X_2X_3X_4$
- Entry phonebook EF Id: X_1X_201

Select $X_1X_2X_3X_4$
Select X_1X_201
Seek for the first empty record, or use another way to find the first empty record "Mobile related".
For example "The first empty record number is 01FF"
Is the record will be hidden? For example "Yes"
Update the found record by putting the name, the principal phone number, the pointer in the creation will point on itself and the value of the pointer will be: 0DFF.

Adding an e-mail address to an existing entry.
Assumptions.
File Id of E-mail file is: X_1X_21F
The phone knows:
The number of the entry phone record
The pointer
The name of the file where the e-mails are stored

Seek for the first empty record in X_1X_21F "e-mail file". For example "The first empty record number is 001A".

If the value of the pointer P1 contained in the entry is pointing to itself.

Then

Update the entry record in EF_{EPB} by making it pointing to the e-mail record "001A" with SFI 1F;

Update the e-mail record by making it pointing to the entry record in EF_{EPB} with SFI 01.

Else

Perform the function F=Read(record) from the file designated by its SFI in P1;

Repeat the same function "F" until the pointer is pointing to the Phonebook Entry file (EF_{EPB}) or to the e-mail file.

Update the e-mail record 001A if the result of the function F is pointing to the Phonebook Entry file; Otherwise Update the e-mail record found by the function F

End If

PERFORMANCE (document 99183)

Card Memory

Memory Optimization

Yes it's possible in the personalization stage.

Memory Consumption for Phone Book Data

Assumptions:

Header	Phone Entry	Nickname	E-mail	Extra Phones
24	33	9	34	35

Every entry has 4 phone numbers linked.

100 entries			
Nickname	0%	50%	100%
E-mail			
0%	7048	7498	7948
30%	8068	8518	8968

200 entries			
Nickname	0%	50%	100%
E-mail			
0%	14048	14948	15848
30%	16088	16988	17888

300 entries			
Nickname	0%	50%	100%
E-mail			
0%	21048	22398	23748
30%	24108	25458	26808

500 entries			
Nickname	0%	50%	100%
E-mail			
0%	35048	37298	39548
30%	40148	42398	44648

Memory Consumption for Phone Book Code (e.g. Operating System, SCQL engine,..)

Not applicable

Access to data linked to entries in the 'Incoming / Outgoing Call' list

It's possible just for the principal phone number.

Adding of new features / elements to the Phone Book

Since the structure is cyclic it's possible to add another attribute as long as the Mobile can manage it. All the length of the record are flexible ,for example we can have just two extra phone instead of 4 ,etc …

Complexity of the implementation (card and mobile)

For the card we need to add another type of file where the number of record is coded in 16 bits, other solutions can be implemented even without creating another type of file.

Access time to Phone Book entries

Time to retrieve all names from the card into the phone

It's faster than 2G phone.

Time for access first matching data after starting a search on the database (Phone Book)

Up to the phone since all the needed data are in the phone.

Time to retrieve linked information of a selected phone book entry

Very fast one APDU due to the format of the pointer in the file.

Error Recovery

We will not have any lost of data or link because for each action we have to send two APDUs first one will be to update the link "reserve the space" and the second one will be to update the data in the corresponding file.

- If any problem happens in the middle of any APDU the existing error recovery will be applied.
- If any problem happens between the two APDUs the structure will be consistent and the data has to be inserted again.