US 20030212896A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2003/0212896 A1

(43) Pub. Date: Nov. 13, 2003

Kisliakov

(54) USER INTERFACE FOR ACCESSING FILES IN A SMARTCARD FILE

(76) Inventor: **Andrew Kisliakov**, Burwood (AU)

Correspondence Address:
FITZPATRICK CELLA HARPER & SCINTO
30 ROCKEFELLER PLAZA
NEW YORK, NY 10112 (US)

(21) Appl. No.: **10/321,507**

(22) Filed: **Dec. 18, 2002**

(30) **Foreign Application Priority Data**

Dec. 20, 2001 (AU) PR9658

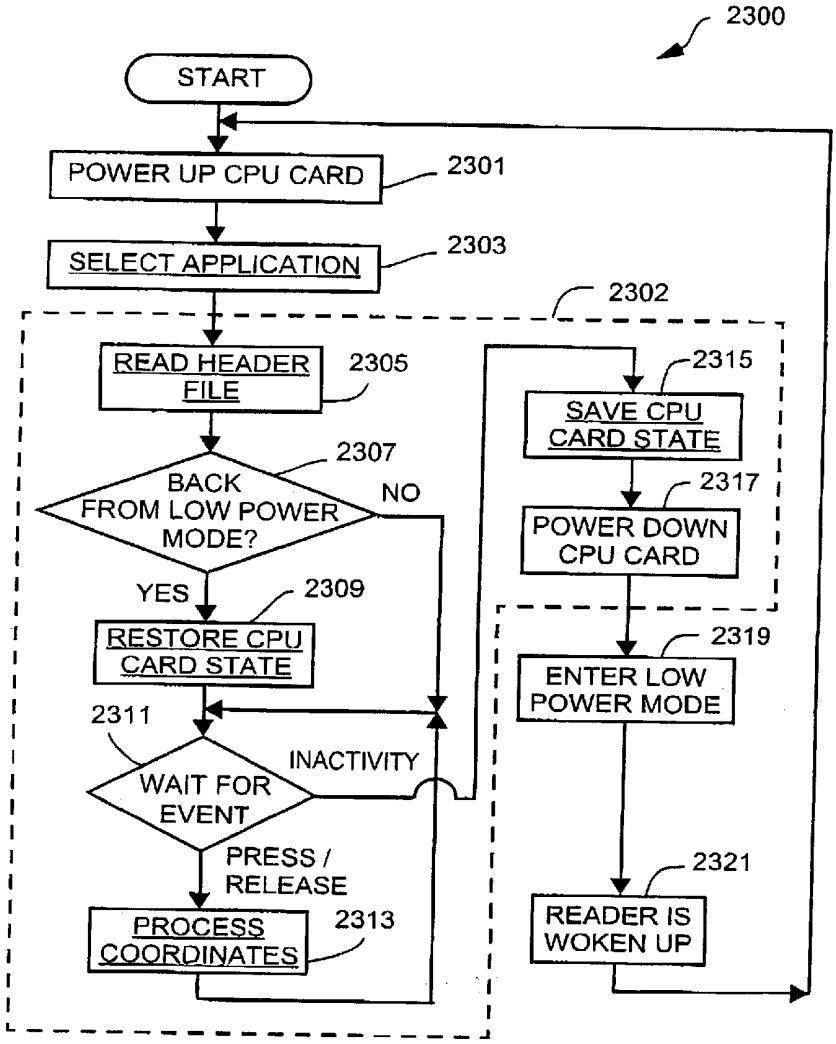
Publication Classification

(51) **Int. Cl.**[7] **G06F 12/14**

(52) **U.S. Cl.** **713/193**; 713/165

(57) **ABSTRACT**

A program, method and apparatus for allowing access to data associated with at least one file stored within an electronic card (**100**) having one or more associated user interface elements (154), is disclosed. The electronic card (**100**) comprises a card portion (**270**) having an electronic apparatus (**259**) attached thereto. The electronic apparatus (**259**) comprises a processor means (**275**) coupled to a memory means (**276**). The processor means (**275**) is adapted for coupling to a reading device (**300**) to facilitate reading of the electronic card (**100**). The memory means (**276**) is configured to retain the files and the program for execution by the processor means (**275**). The program relates reading signals generated from a user selection of at least one of the associated user interface elements (**154**) with a first file stored within the memory (**276**). The first file is classified by the program as protected. The program allows access to data of a second file associated with the first file. The second file is classified by the program as unprotected.



APPLE 1010

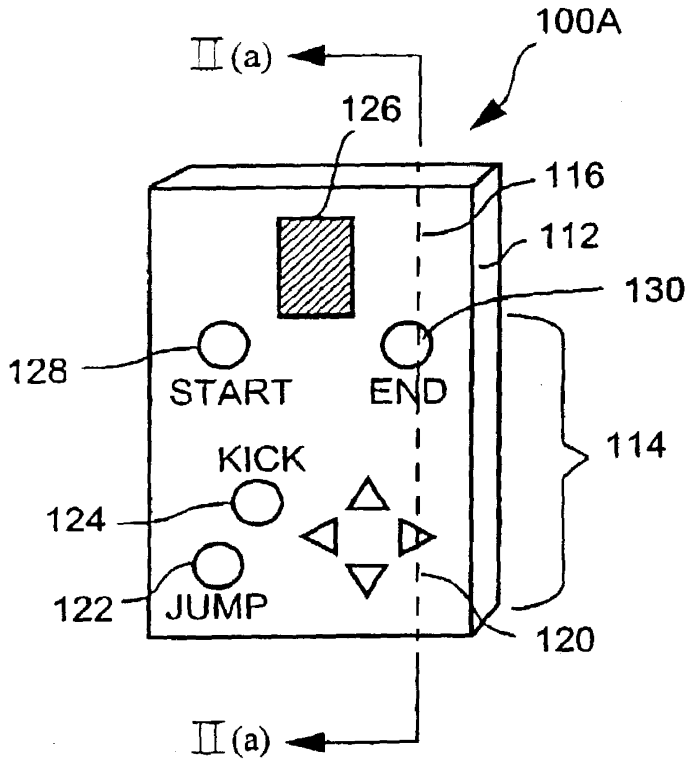

100A
II (a)
126
116
112
130
128
START
END
114
KICK
124
122
JUMP
120
II (a)

Fig. 1(a)

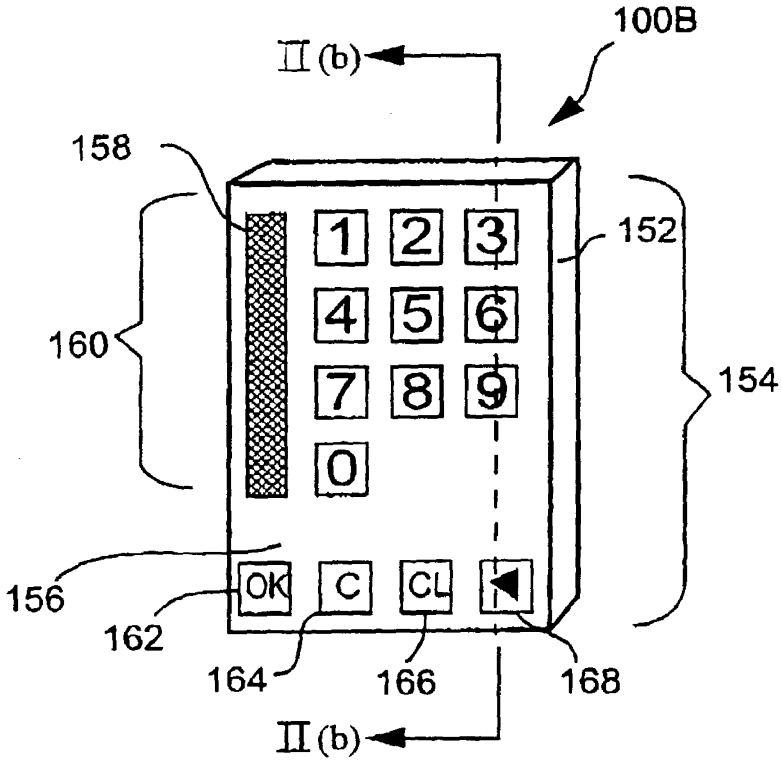

100B
II (b)
158
152
1
2
3
4
5
6
160
7
8
9
154
0
156
OK
C
CL
162
164
166
168
II (b)

Fig. 1(b)

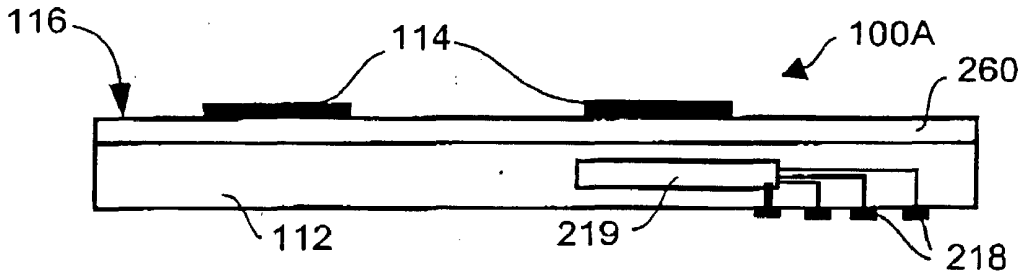

116
114
100A
260
112
219
218

Fig. 2(a)

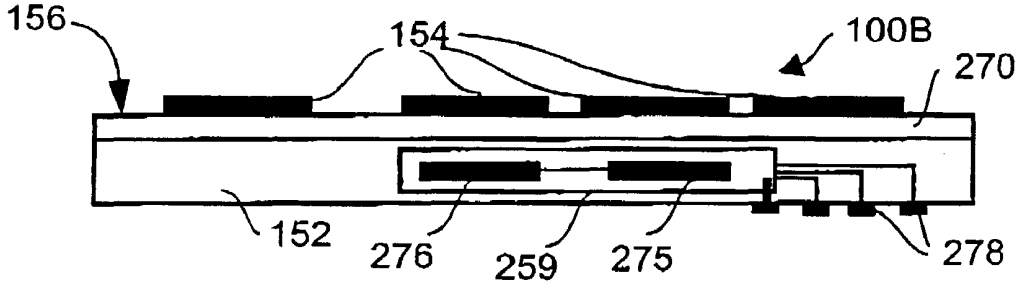

156
154
100B
270
152
276
259
275
278

Fig. 2(b)

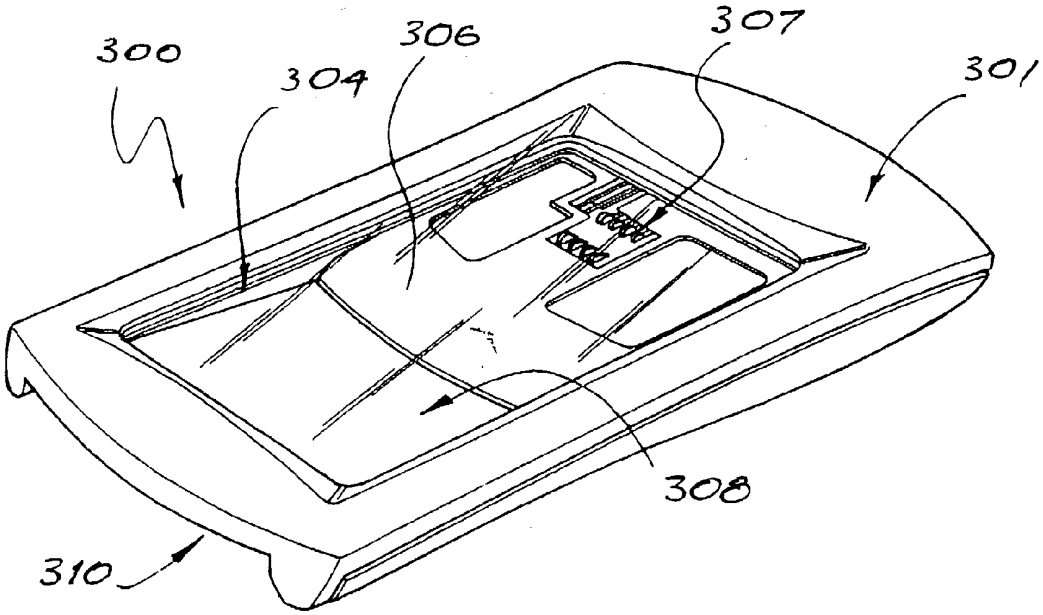


FIG. 3

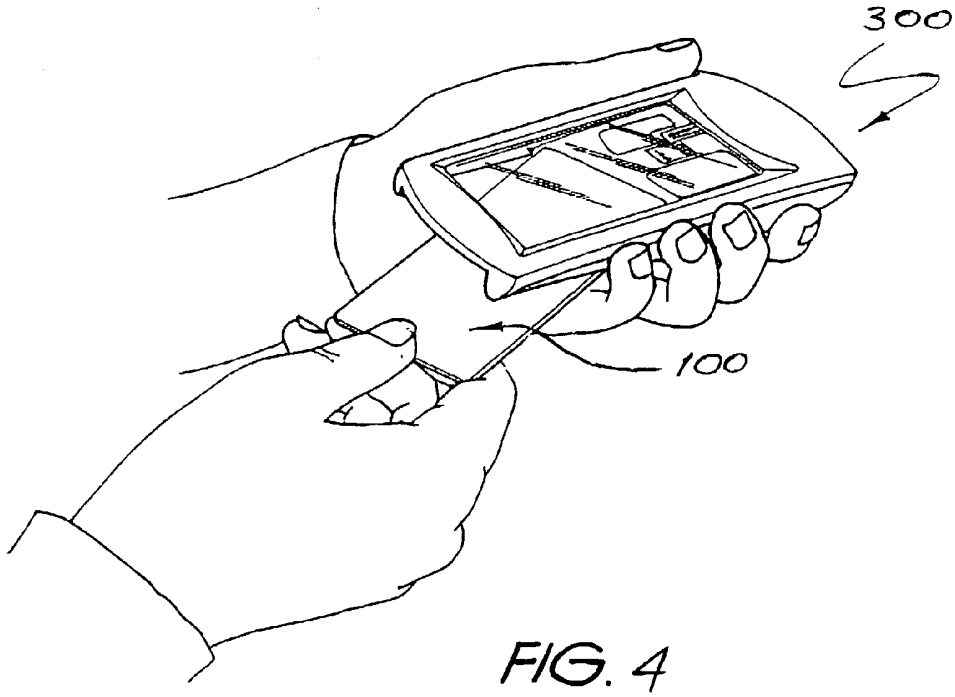


FIG. 4

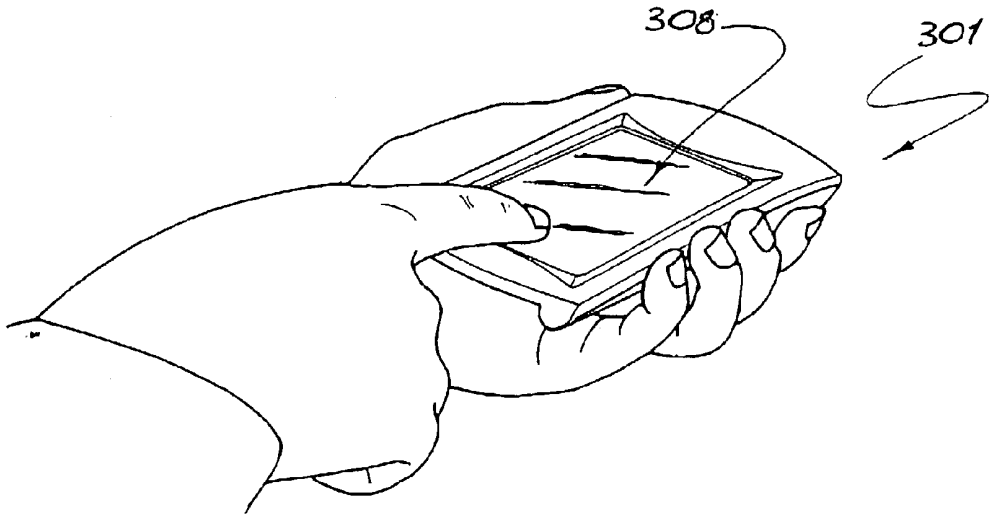


FIG. 5

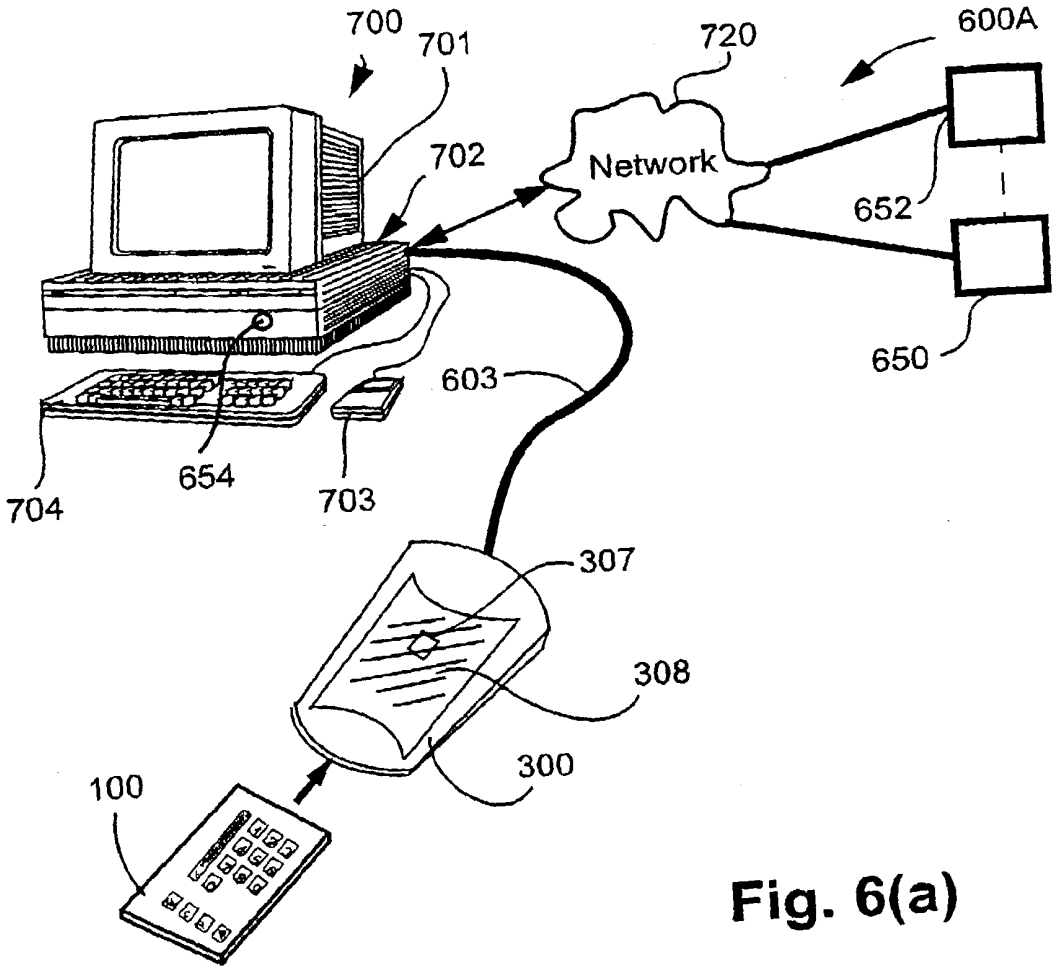

700
701
720
600A
702
Network
652
650
603
654
704
703
307
308
300
100

Fig. 6(a)

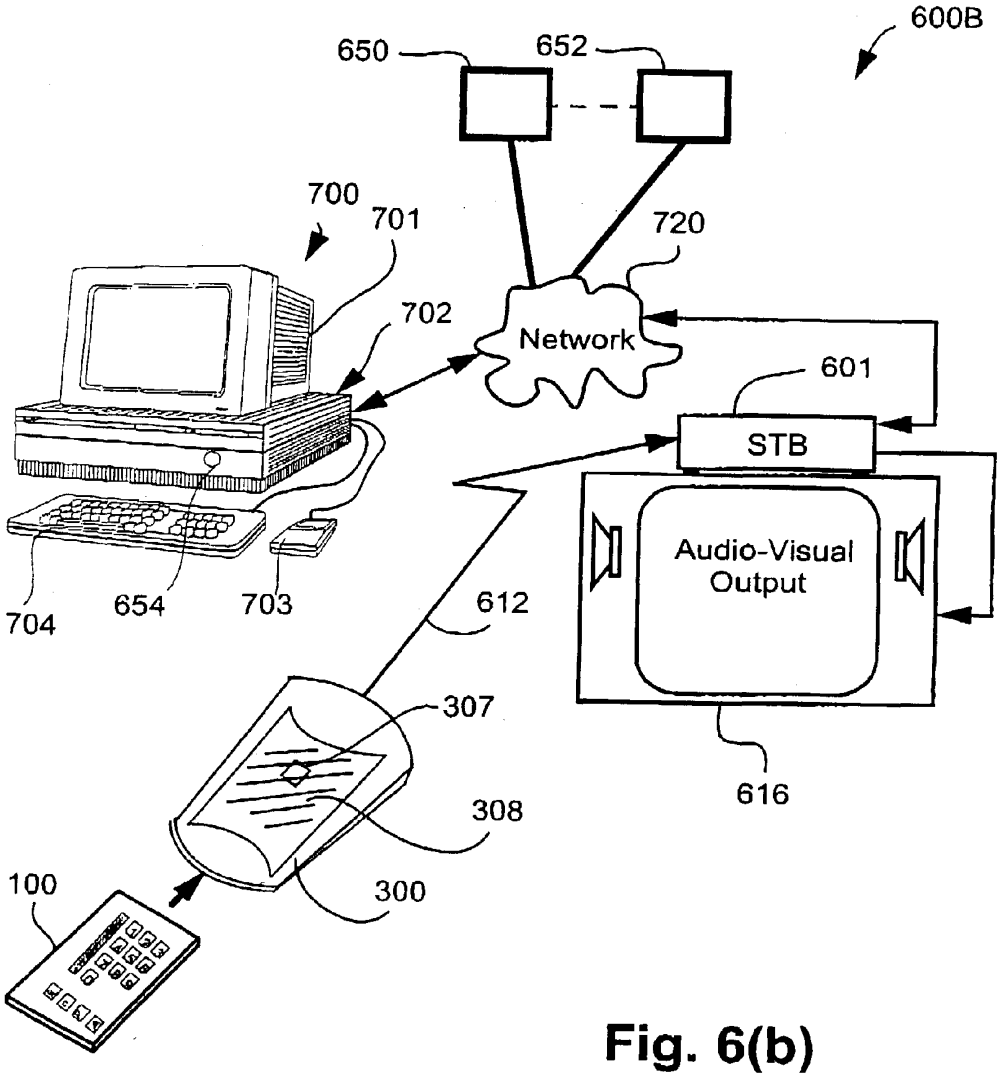

600B
650
652
700
701
720
702
Network
601
STB
Audio-Visual
Output
704
654
703
612
307
616
308
100
300

Fig. 6(b)

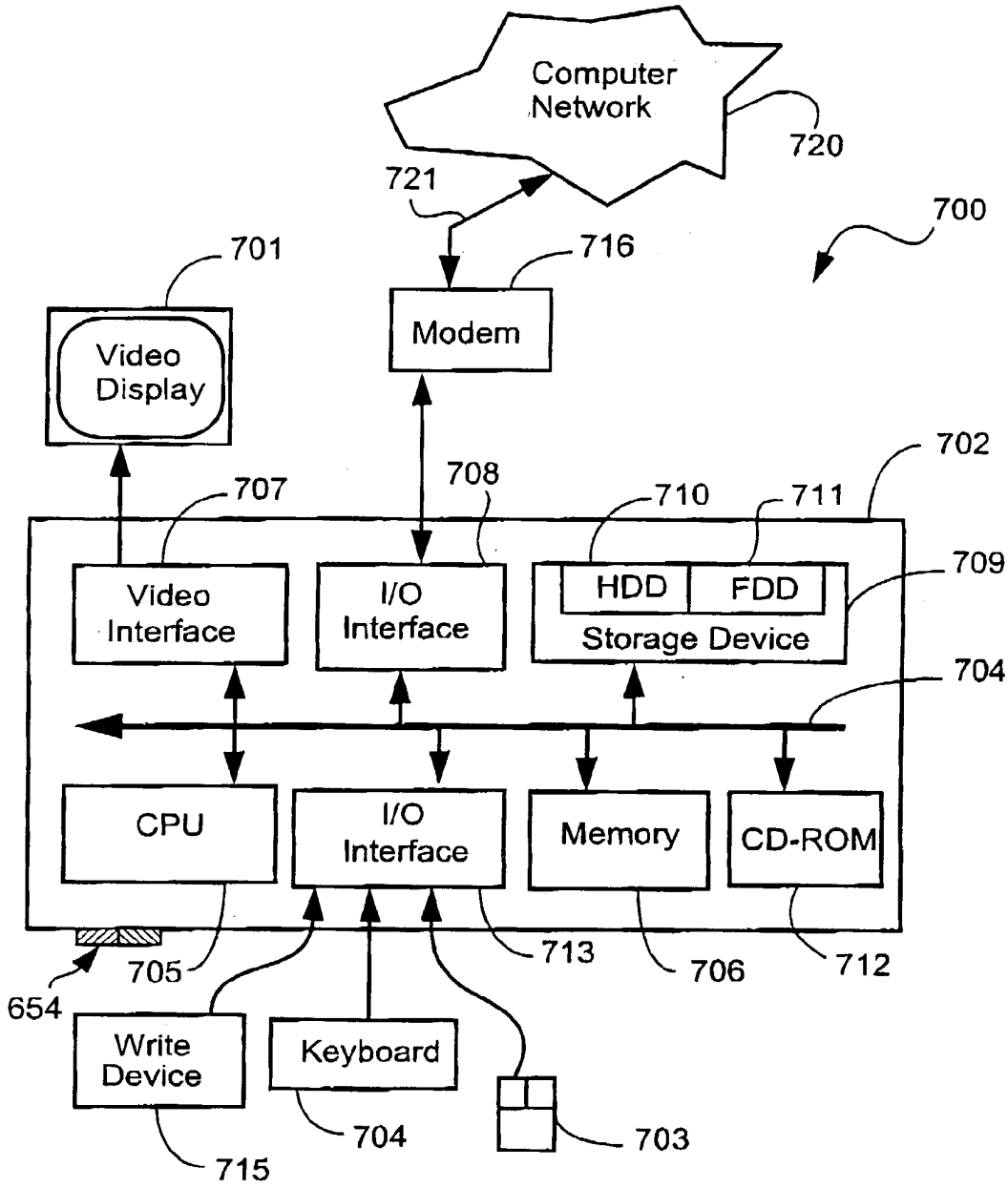

Computer Network
720
721
700
701
716
Video Display
Modem
702
707
708
710
711
709
Video Interface
I/O Interface
HDD
FDD
Storage Device
704
CPU
I/O Interface
Memory
CD-ROM
654
705
713
706
712
Write Device
Keyboard
704
703
715

Fig. 7

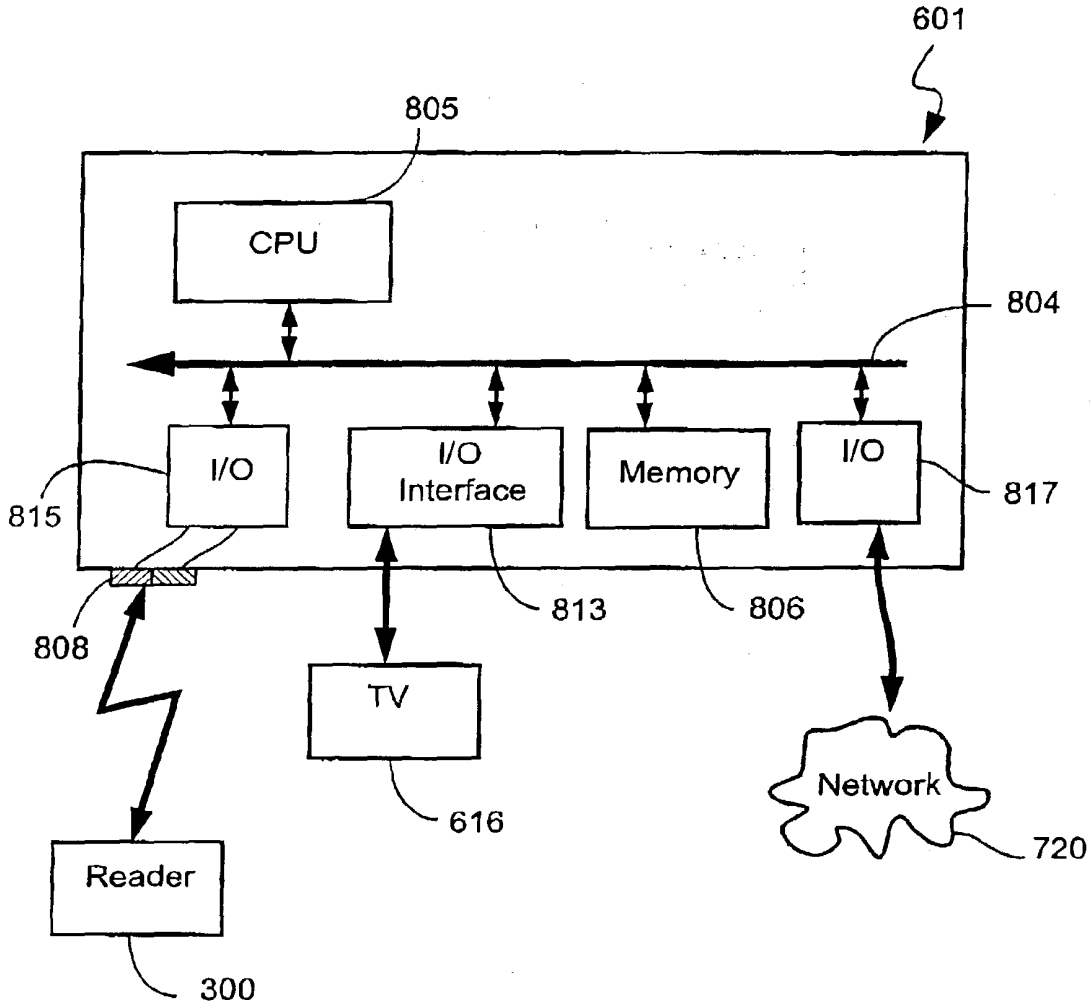

601
805
CPU
804
I/O
I/O
Interface
Memory
I/O
815
817
808
813
806
TV
Network
616
720
Reader
300

Fig. 8

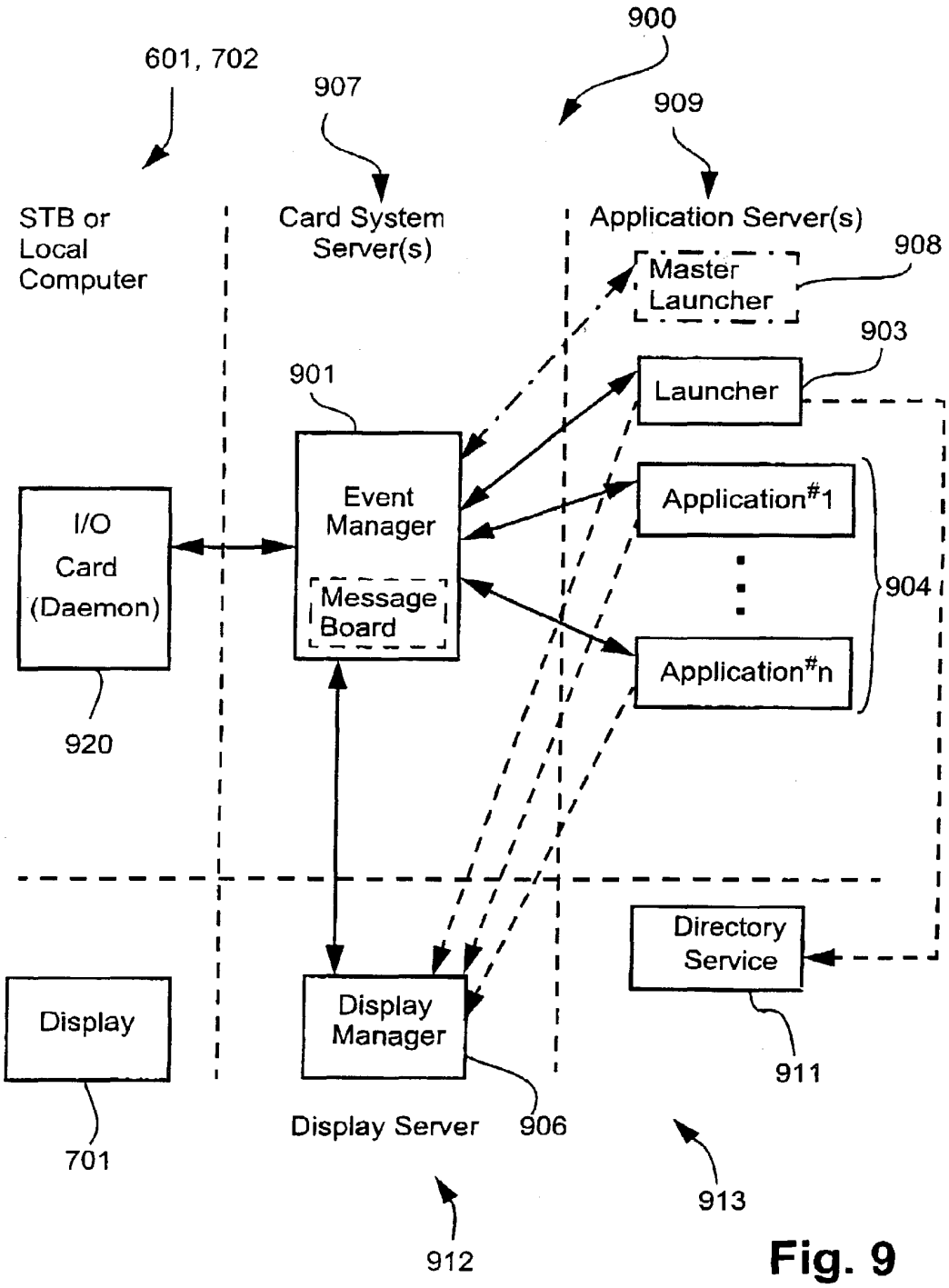

900
601, 702
907
909
STB or Local Computer
Card System Server(s)
Application Server(s)
908
Master Launcher
903
901
Launcher
Event Manager
Message Board
Application#1
I/O Card (Daemon)
904
Application#n
920
Directory Service
Display
Display Manager
911
701
Display Server
906
913
912

Fig. 9

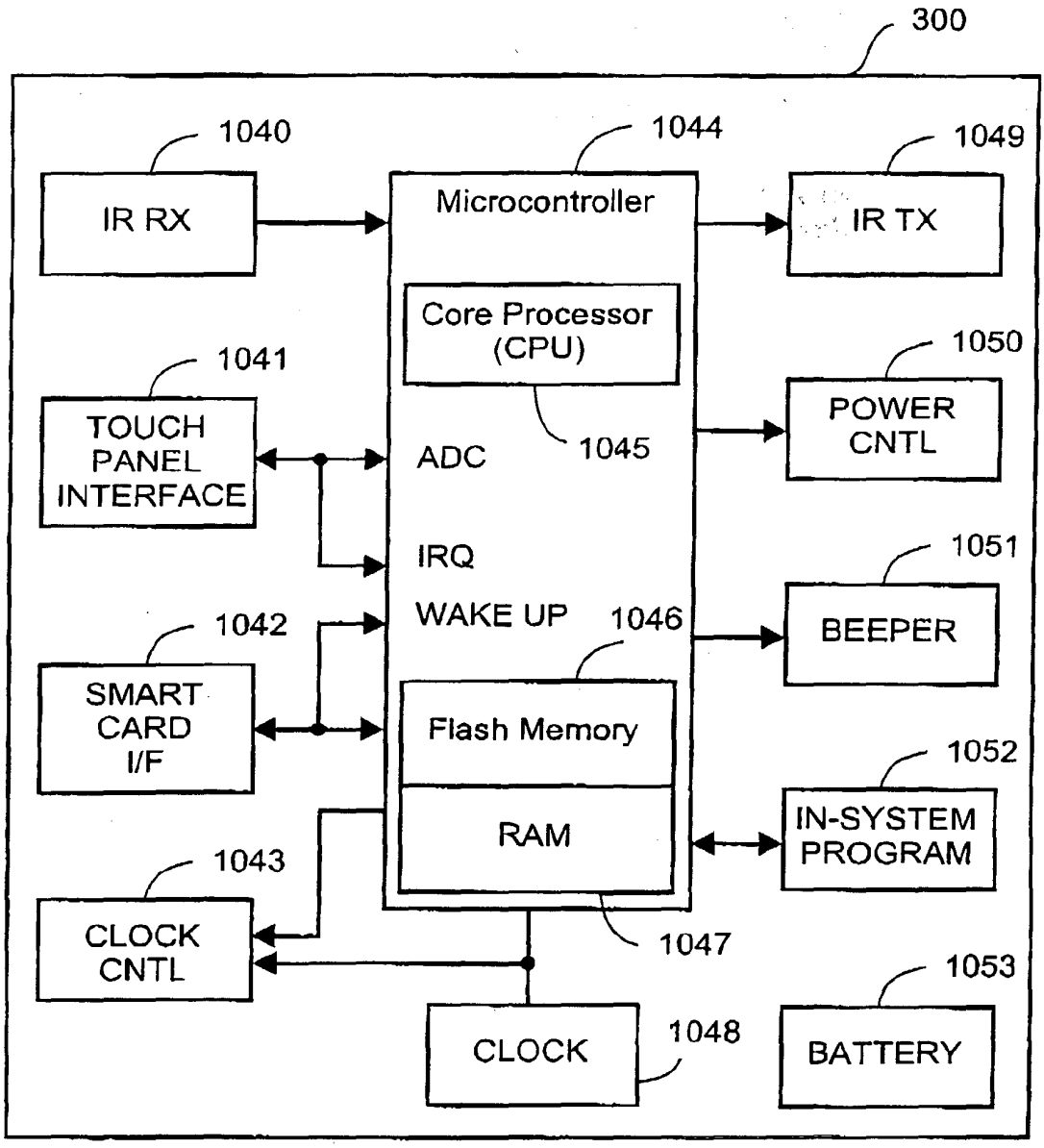

300
1040
IR RX
1044
Microcontroller
1049
IR TX
Core Processor
(CPU)
1041
TOUCH
PANEL
INTERFACE
ADC
1045
1050
POWER
CNTL
IRQ
1051
WAKE UP
1046
BEEPER
1042
SMART
CARD
I/F
Flash Memory
1052
RAM
IN-SYSTEM
PROGRAM
1043
CLOCK
CNTL
1047
1053
CLOCK
1048
BATTERY

Fig. 10

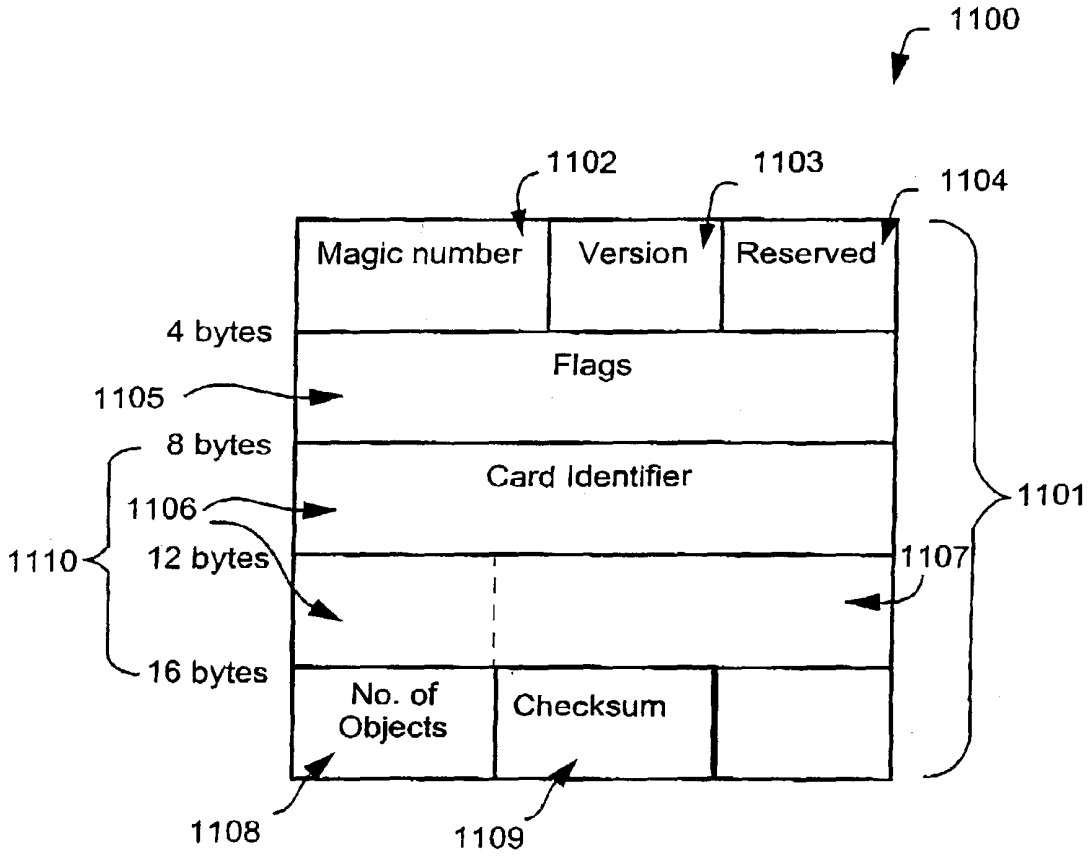

1100
1102
1103
1104
Magic number
Version
Reserved
4 bytes
Flags
1105
8 bytes
Card Identifier
1106
1101
1110
12 bytes
1107
16 bytes
No. of Objects
Checksum
1108
1109

Fig. 11

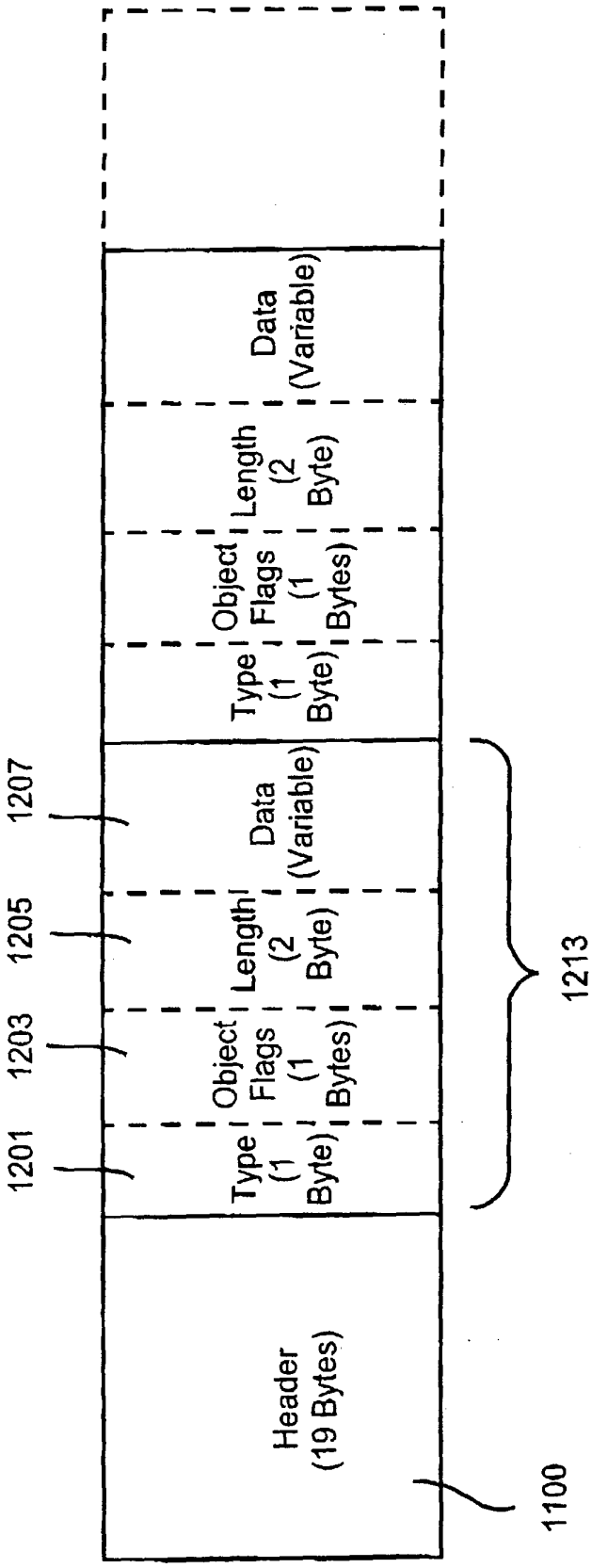

Header
(19 Bytes)
1100
1201
Type
(1
Byte)
1203
Object
Flags
(1
Bytes)
1205
Length
(2
Byte)
1207
Data
(Variable)
1213
Type
(1
Byte)
Object
Flags
(1
Bytes)
Length
(2
Byte)
Data
(Variable)

Fig. 12

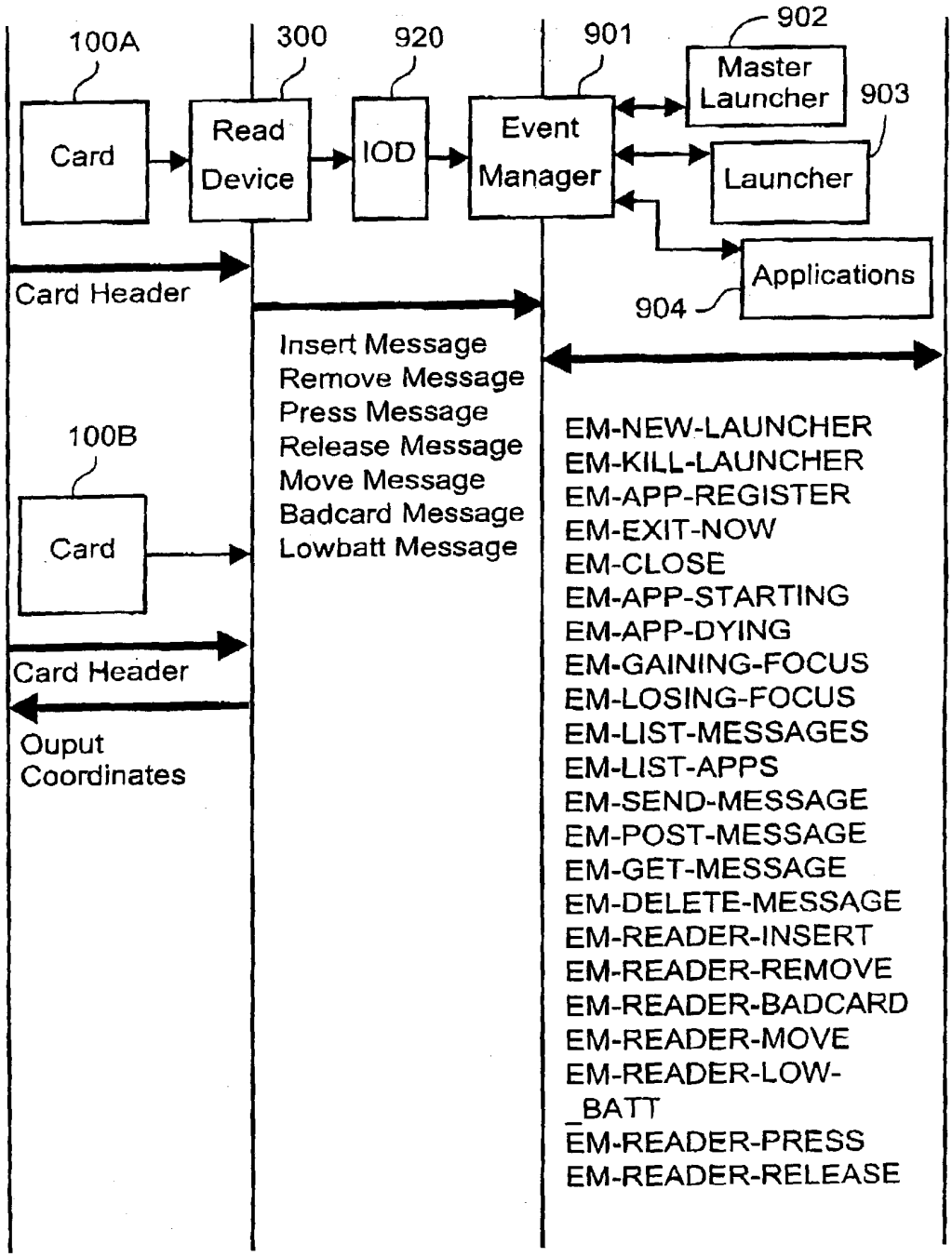

100A
300
920
901
902
Card
Read Device
IOD
Event Manager
Master Launcher
903
Launcher
Applications
904
Card Header
Insert Message
Remove Message
Press Message
Release Message
Move Message
Badcard Message
Lowbatt Message
100B
Card
Card Header
Ouput Coordinates
EM-NEW-LAUNCHER
EM-KILL-LAUNCHER
EM-APP-REGISTER
EM-EXIT-NOW
EM-CLOSE
EM-APP-STARTING
EM-APP-DYING
EM-GAINING-FOCUS
EM-LOSING-FOCUS
EM-LIST-MESSAGES
EM-LIST-APPS
EM-SEND-MESSAGE
EM-POST-MESSAGE
EM-GET-MESSAGE
EM-DELETE-MESSAGE
EM-READER-INSERT
EM-READER-REMOVE
EM-READER-BADCARD
EM-READER-MOVE
EM-READER-LOW-
_BATT
EM-READER-PRESS
EM-READER-RELEASE

Fig. 13

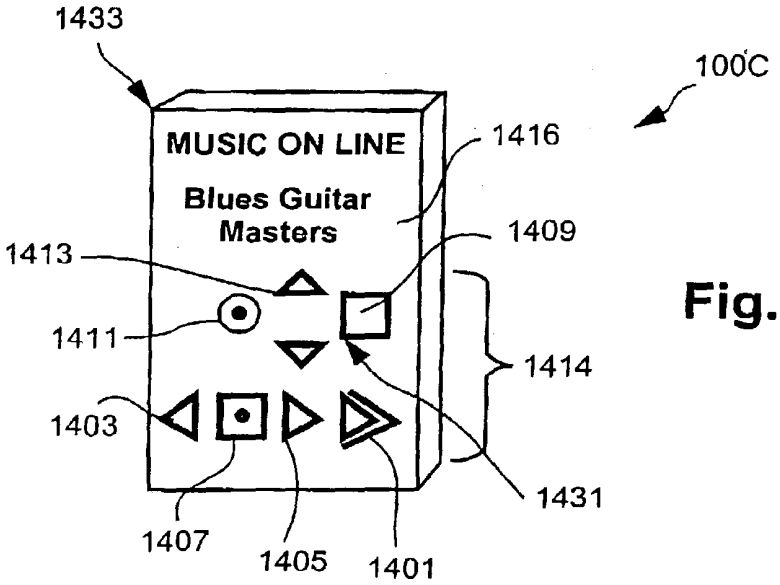


Fig. 14(a)

Fig. 14(b)

User Interface Element	User Interface Element Object Fields					
	Flags	X1	Y1	X2	Y2	Data
△	0x20	0005	0005	0006	0006	xxxx
▽	0x20	0005	0007	0006	0008	xxxx
⊙	0x20	0003	0006	0004	0007	xxxx
⊡	0x20	0003	0008	0004	0009	xxxx
▷	0x20	0005	0008	0006	0009	xxxx
◁	0x20	0001	0008	0002	0009	xxxx
▷▷	0x20	0007	0008	0008	0009	xxxx
□	0x20	0007	0006	0008	0007	0015

1409 1420 1421 1422 1431 1423 1424 1425

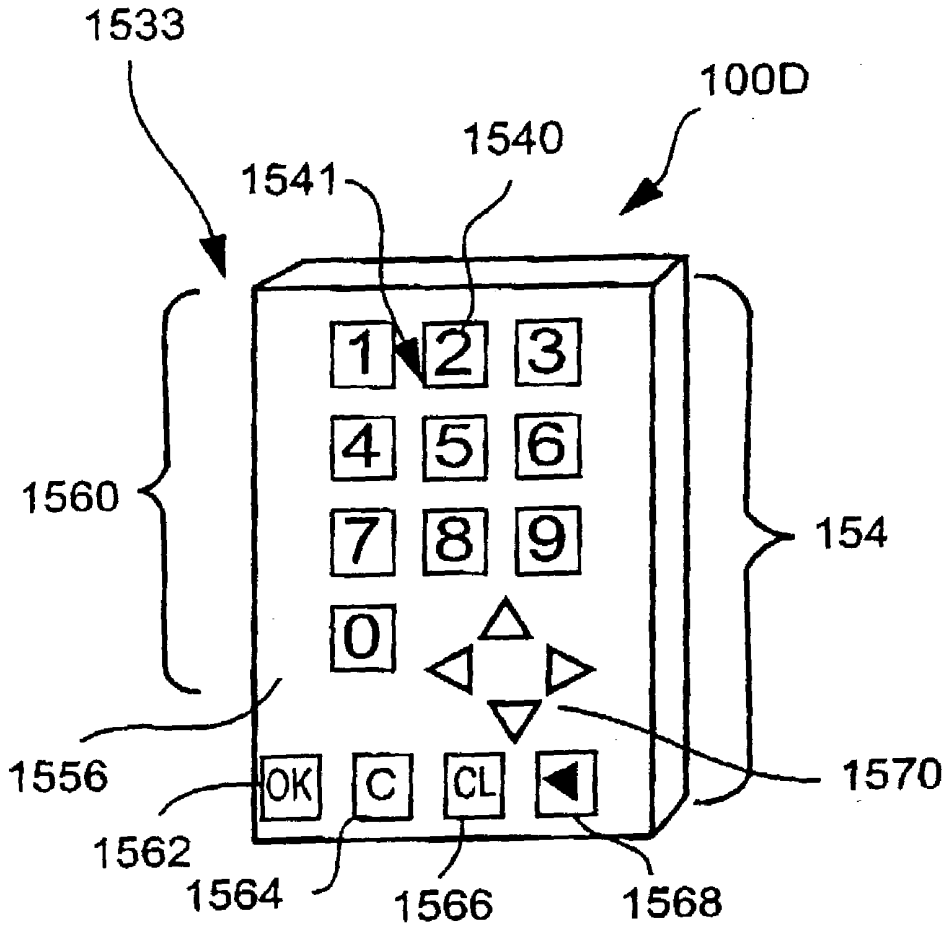

1533
100D
1540
1541
1560
154
1556
1570
1562
1564
1566
1568
1
2
3
4
5
6
7
8
9
0
OK
C
CL

Fig. 15(a)

Fig. 15(b)

1531

1540 1520 1521 1522 1523 1524 1525

User Interface Element	User Interface Element Object Fields					
	Flags	X1	Y1	X2	Y2	Data
1	0x20	0003	0001	0004	0002	xxxx
2	0x20	0005	0001	0006	0002	0032
3	0x20	0007	0001	0008	0002	xxxx
4	0x20	0003	0003	0004	0004	xxxx
5	0x20	0005	0003	0006	0004	xxxx
6	0x20	0007	0003	0008	0004	xxxx
7	0x20	0003	0004	0004	0005	xxxx
8	0x20	0005	0004	0006	0005	xxxx
9	0x20	0007	0004	0008	0005	xxxx
0	0x20	0003	0006	0004	0007	xxxx
OK	0x20	0001	0008	0002	000A	xxxx
C	0x20	0003	0008	0004	000A	xxxx
CL	0x20	0005	0008	0006	000A	xxxx
◀	0x20	0007	0008	0008	000A	xxxx
▷	0x20	0007	0006	0008	0007	xxxx
◁	0x20	0005	0006	0006	0007	xxxx
△	0x20	0006	0005	0007	0006	xxxx
▽	0x20	0006	0007	0007	0008	xxxx

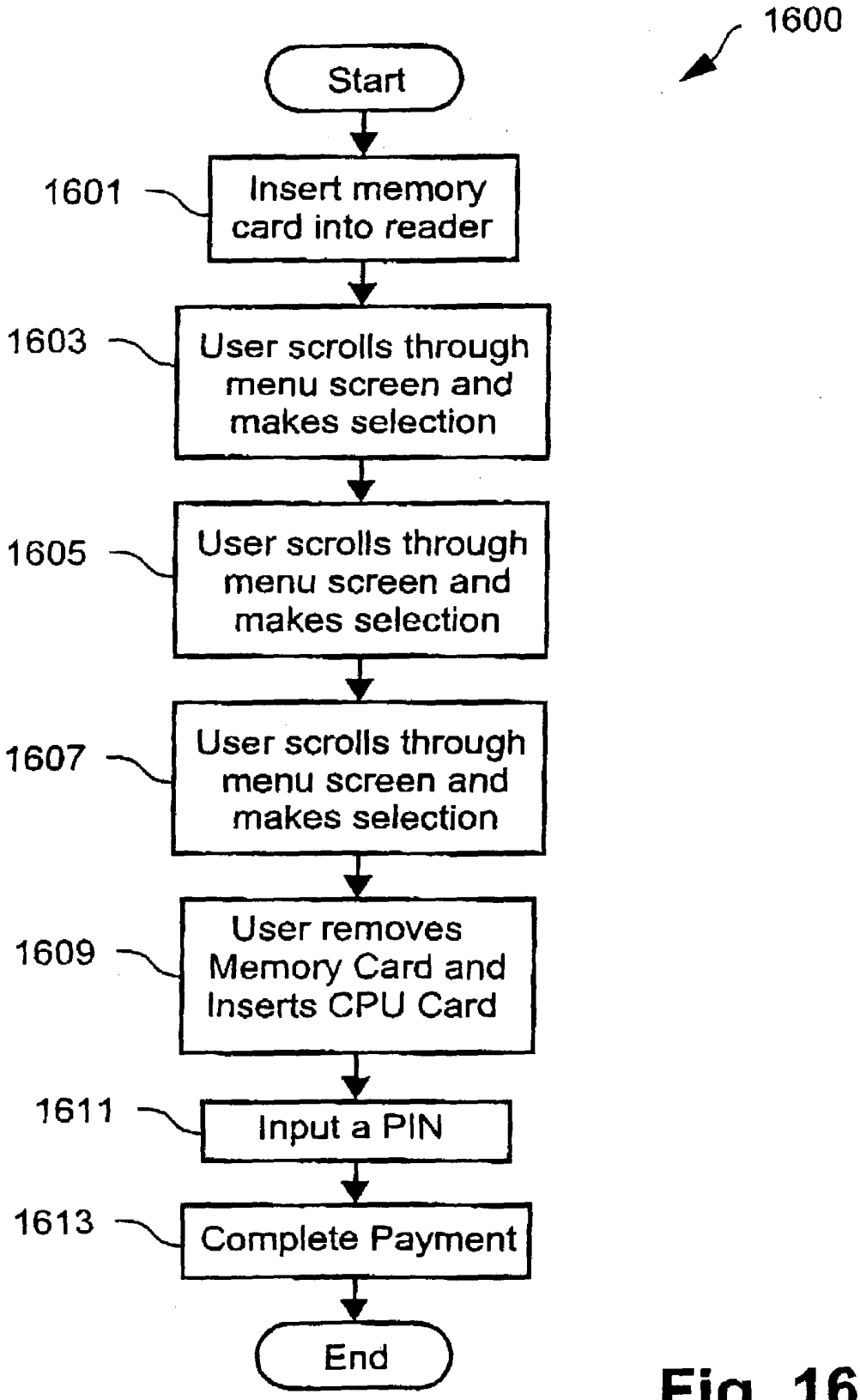

1600
Start
1601
Insert memory card into reader
1603
User scrolls through menu screen and makes selection
1605
User scrolls through menu screen and makes selection
1607
User scrolls through menu screen and makes selection
1609
User removes Memory Card and Inserts CPU Card
1611
Input a PIN
1613
Complete Payment
End

Fig. 16

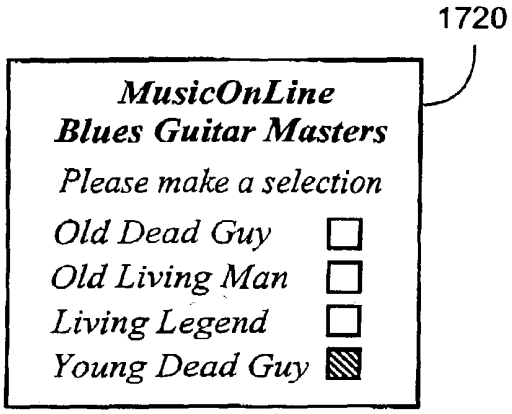


Fig. 17(a)

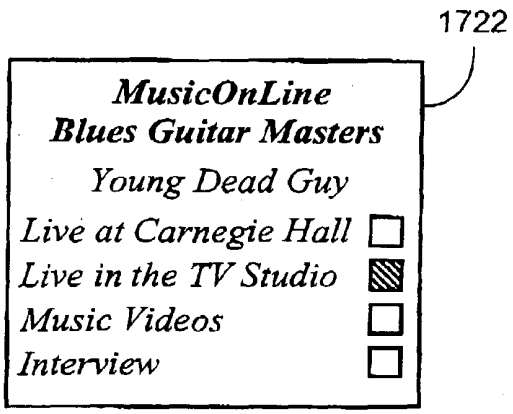


Fig. 17(b)

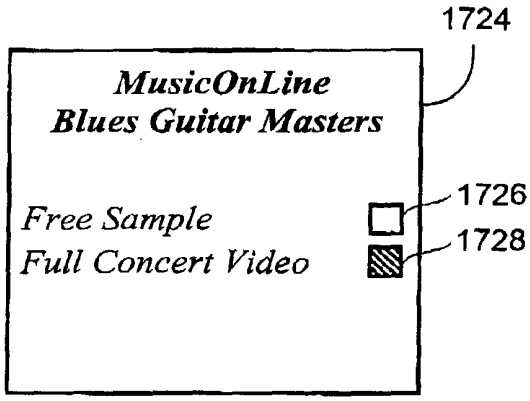


Fig. 17(c)

Fig. 17(d)

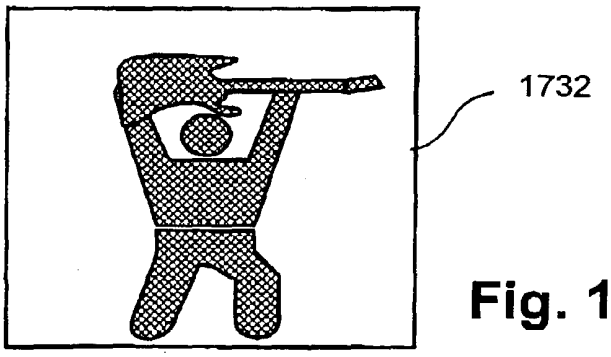


Fig. 17(e)

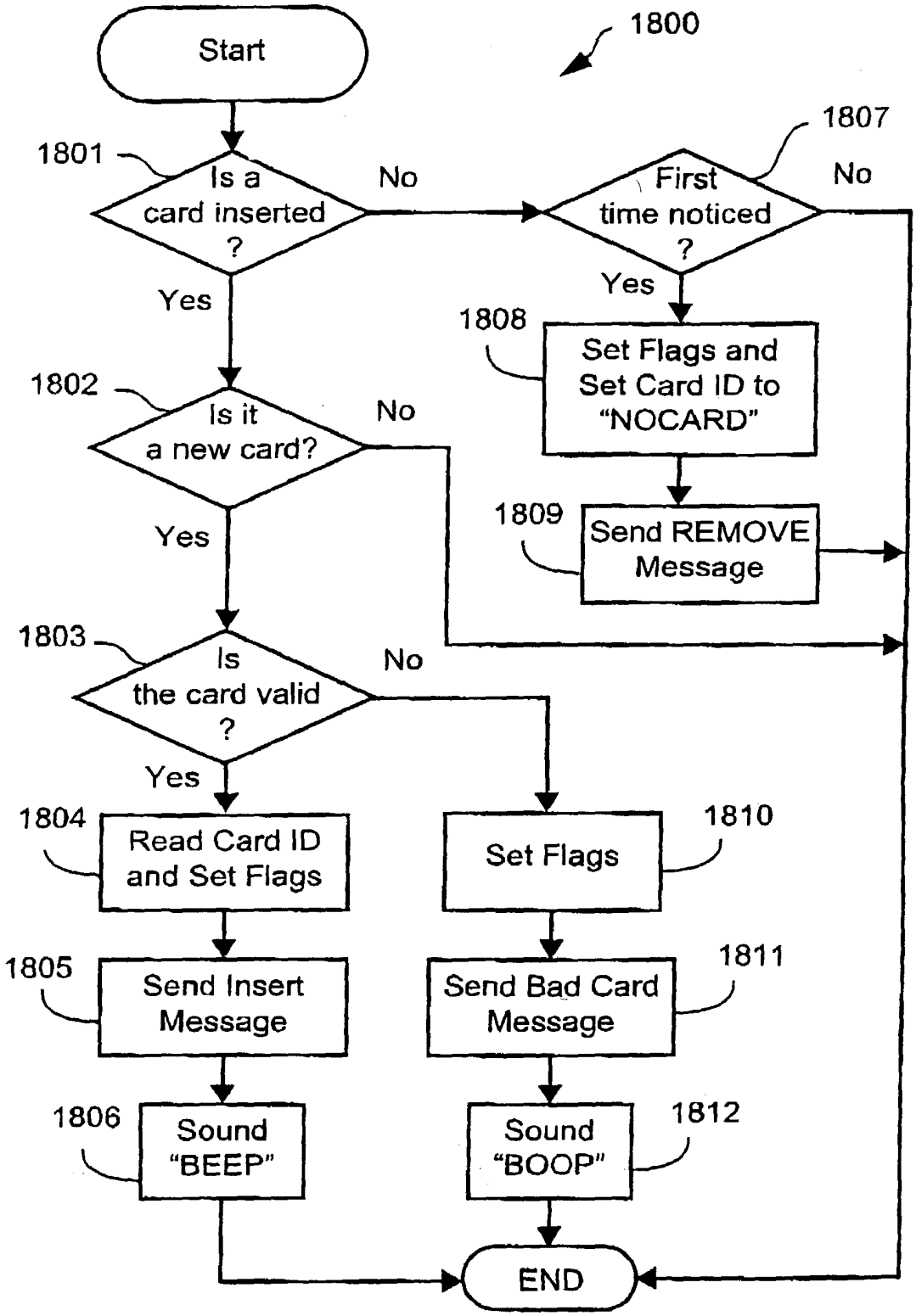

1800
Start
1801
Is a card inserted ?
No
1807
First time noticed ?
No
Yes
Yes
1808
Set Flags and Set Card ID to "NOCARD"
1802
Is it a new card?
No
1809
Send REMOVE Message
Yes
1803
Is the card valid ?
No
Yes
1804
Read Card ID and Set Flags
Set Flags
1810
1805
Send Insert Message
Send Bad Card Message
1811
1806
Sound "BEEP"
Sound "BOOP"
1812
END

Fig. 18

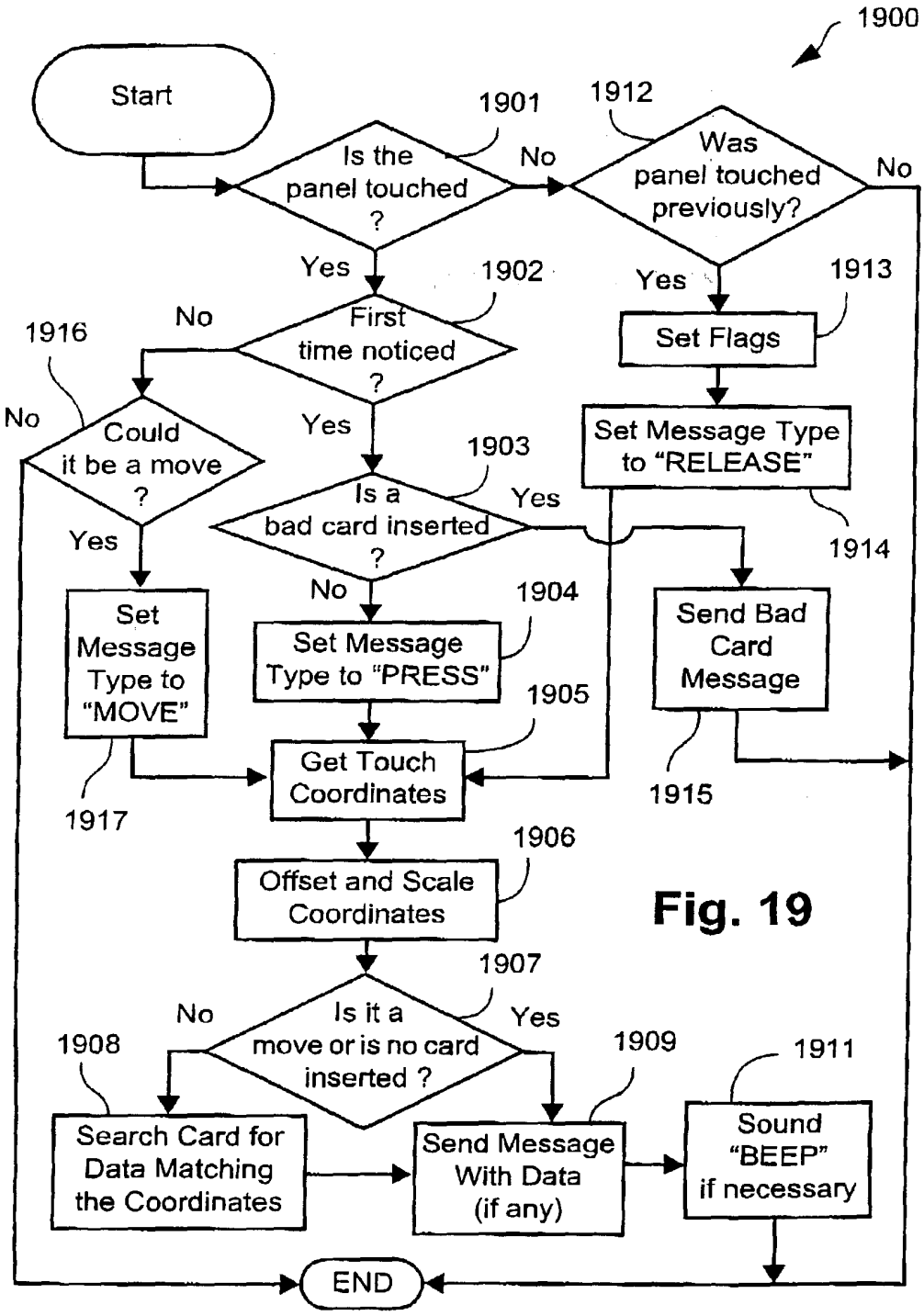

1900
Start
1901
Is the panel touched ?
No
1912
Was panel touched previously?
No
Yes
1902
First time noticed ?
Yes
1913
Set Flags
No
1916
Could it be a move ?
No
Yes
1903
Is a bad card inserted ?
Yes
Set Message Type to "RELEASE"
1914
Yes
Set Message Type to "MOVE"
No
1904
Set Message Type to "PRESS"
Send Bad Card Message
1905
Get Touch Coordinates
1917
1915
1906
Offset and Scale Coordinates
1907
Is it a move or is no card inserted ?
No
Yes
1908
1909
1911
Search Card for Data Matching the Coordinates
Send Message With Data (if any)
Sound "BEEP" if necessary
END

Fig. 19

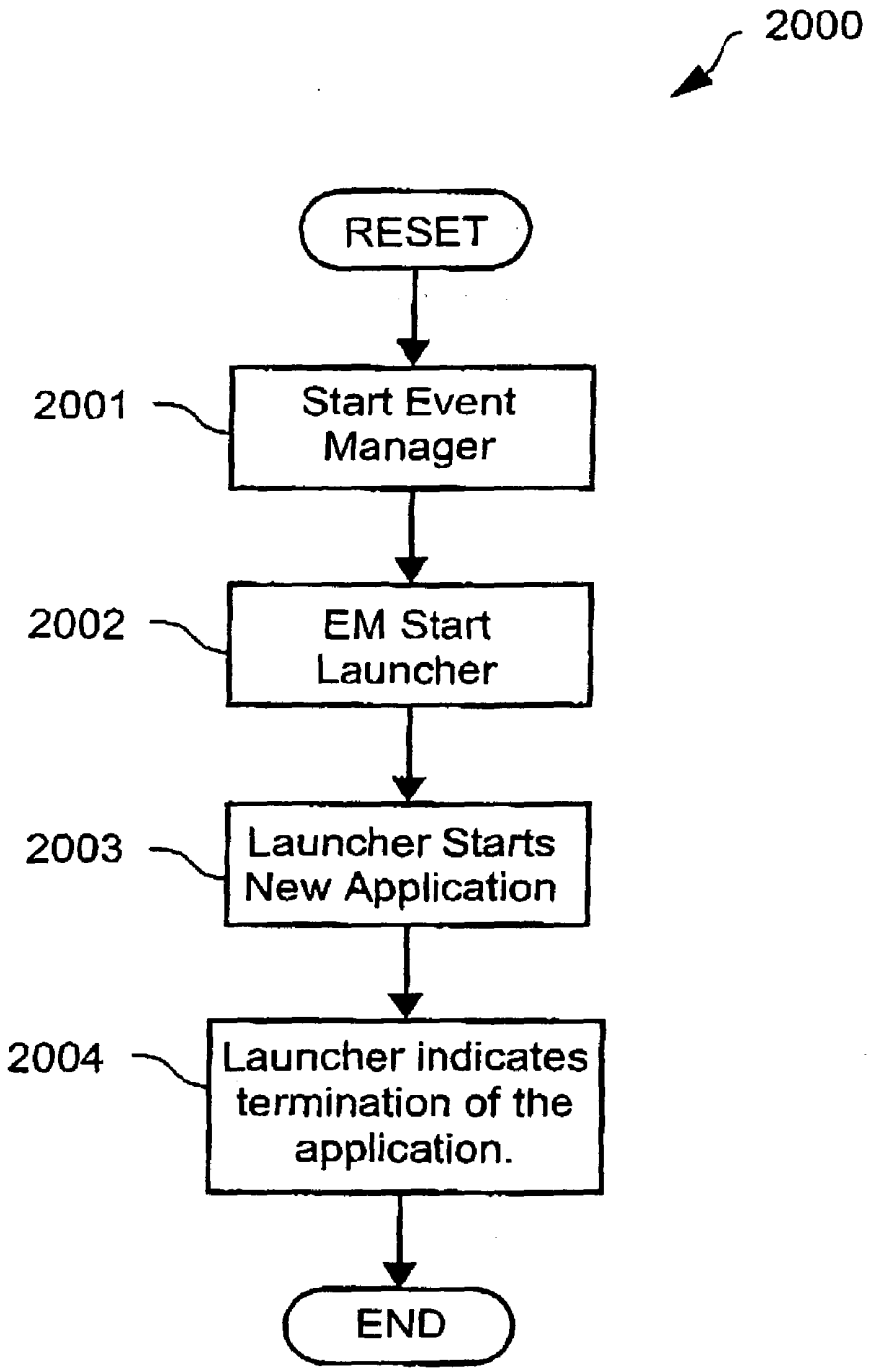

2000
RESET
2001
Start Event Manager
2002
EM Start Launcher
2003
Launcher Starts New Application
2004
Launcher indicates termination of the application.
END

Fig. 20

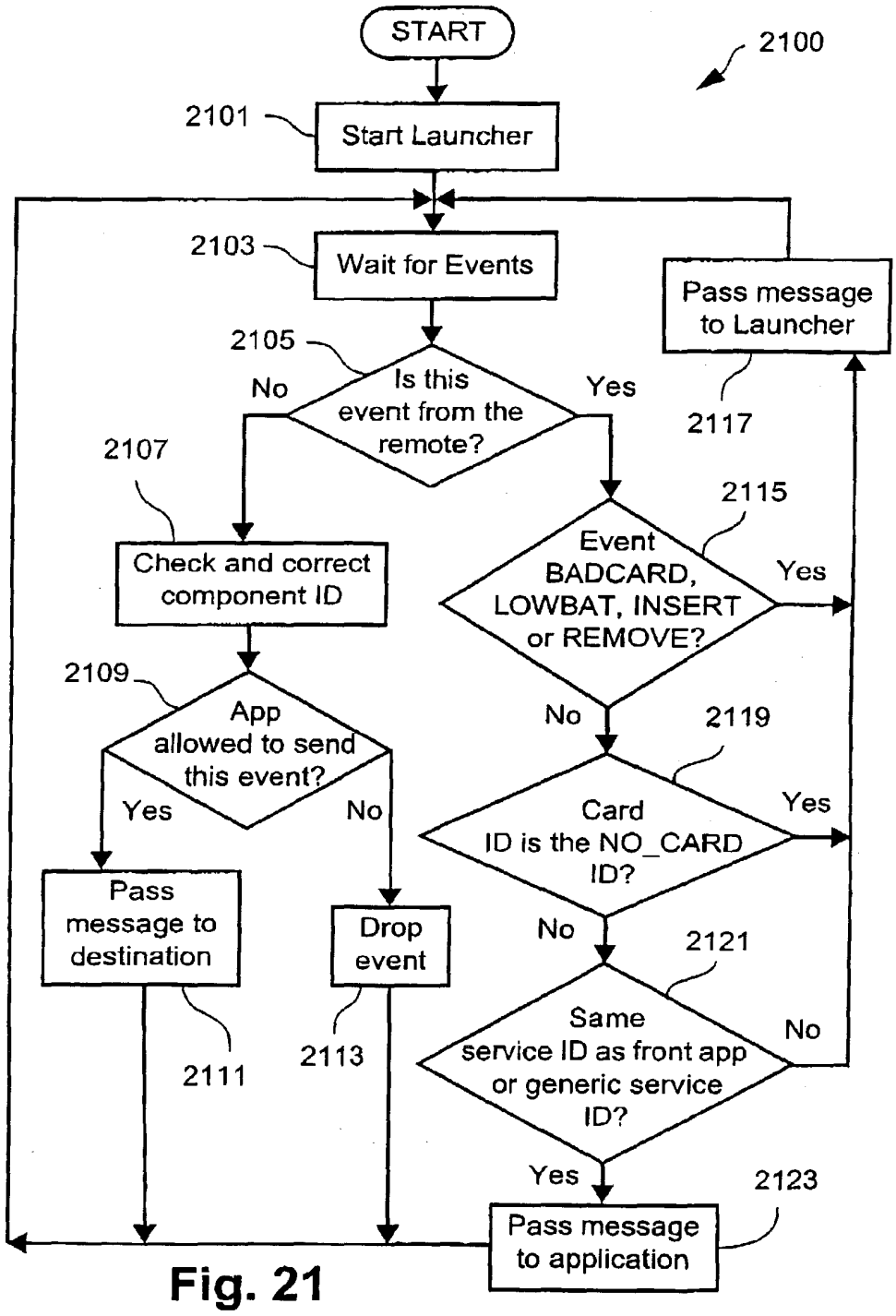

START
2100
2101
Start Launcher
2103
Wait for Events
Pass message to Launcher
2105
Is this event from the remote?
No
Yes
2117
2107
2115
Check and correct component ID
Event BADCARD, LOWBAT, INSERT or REMOVE?
Yes
2109
App allowed to send this event?
No
2119
Yes
No
Card ID is the NO_CARD ID?
Yes
Pass message to destination
Drop event
No
2121
2111
2113
Same service ID as front app or generic service ID?
No
Yes
2123
Pass message to application

Fig. 21

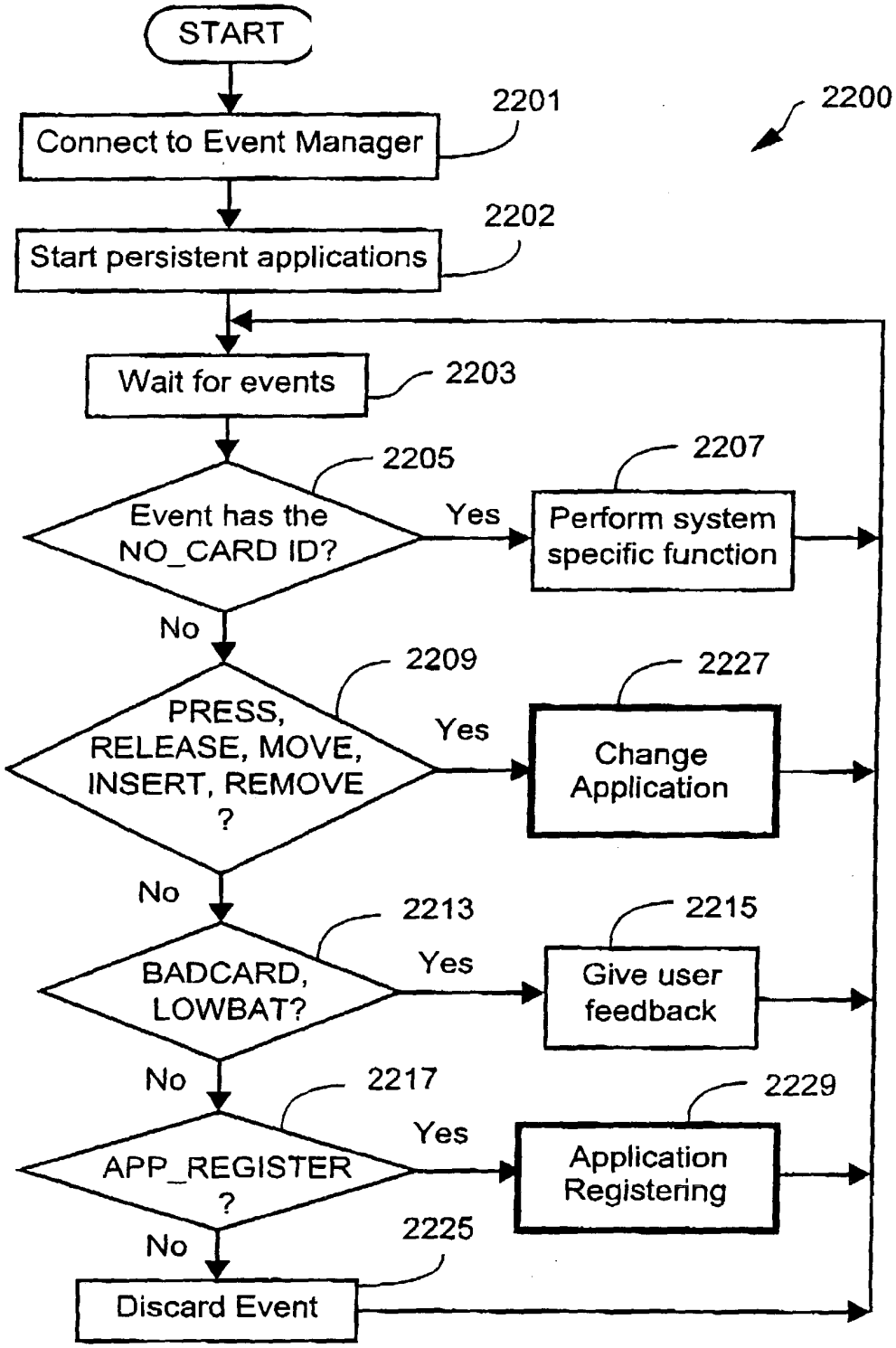

START
Connect to Event Manager
2201
2200
Start persistent applications
2202
Wait for events
2203
2205
Event has the NO_CARD ID?
Yes
2207
Perform system specific function
No
2209
PRESS, RELEASE, MOVE, INSERT, REMOVE ?
Yes
2227
Change Application
No
2213
BADCARD, LOWBAT?
Yes
2215
Give user feedback
No
2217
APP_REGISTER ?
Yes
2229
Application Registering
No
2225
Discard Event

Fig. 22

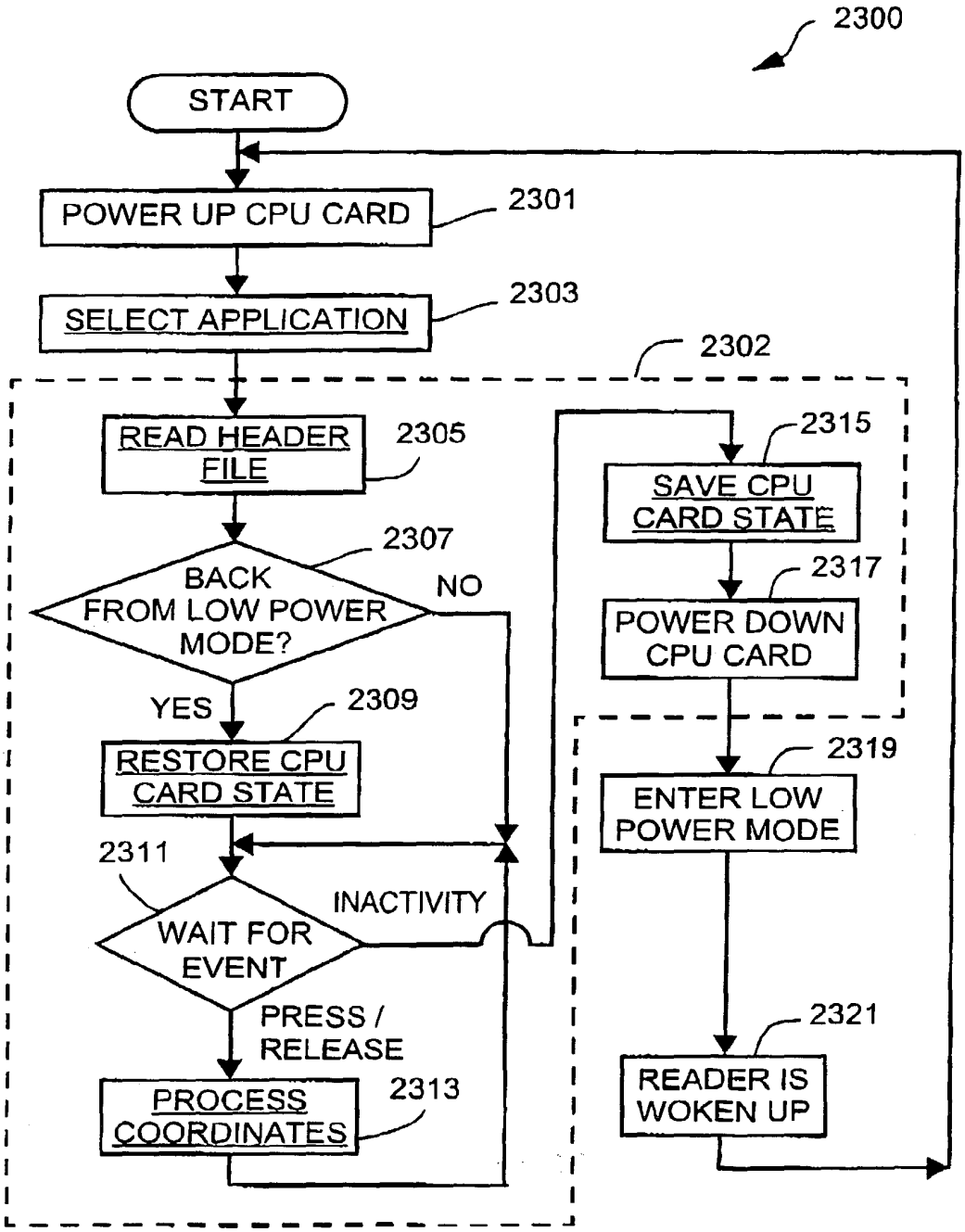

2300
START
POWER UP CPU CARD
2301
SELECT APPLICATION
2303
2302
READ HEADER FILE
2305
2307
BACK FROM LOW POWER MODE?
NO
YES
2309
RESTORE CPU CARD STATE
2311
WAIT FOR EVENT
INACTIVITY
PRESS / RELEASE
PROCESS COORDINATES
2313
2315
SAVE CPU CARD STATE
2317
POWER DOWN CPU CARD
2319
ENTER LOW POWER MODE
2321
READER IS WOKEN UP

Fig. 23

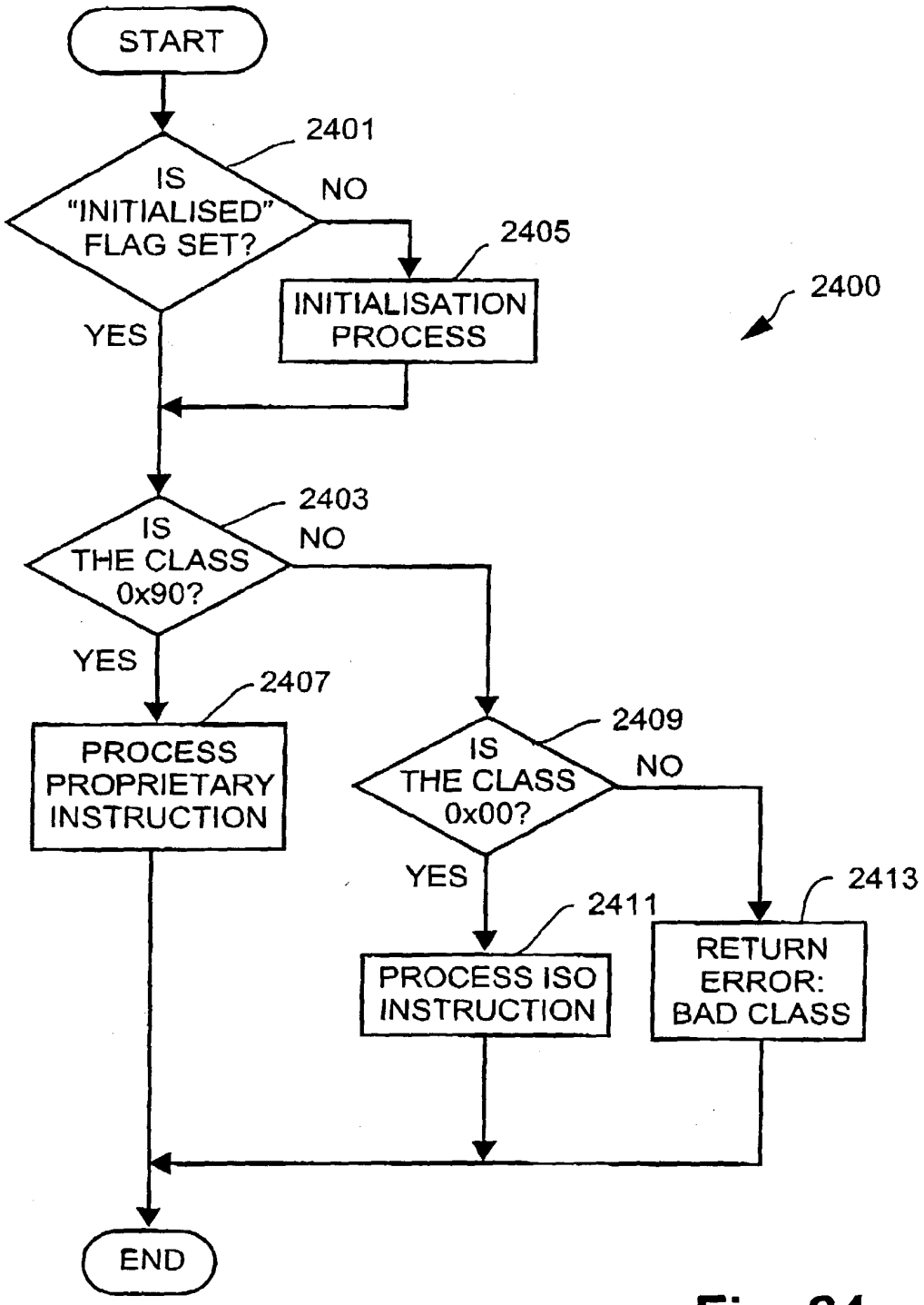

START
2401
IS "INITIALISED" FLAG SET?
NO
2405
INITIALISATION PROCESS
2400
YES
2403
IS THE CLASS 0x90?
NO
YES
2407
PROCESS PROPRIETARY INSTRUCTION
2409
IS THE CLASS 0x00?
NO
YES
2411
PROCESS ISO INSTRUCTION
2413
RETURN ERROR: BAD CLASS
END

Fig. 24

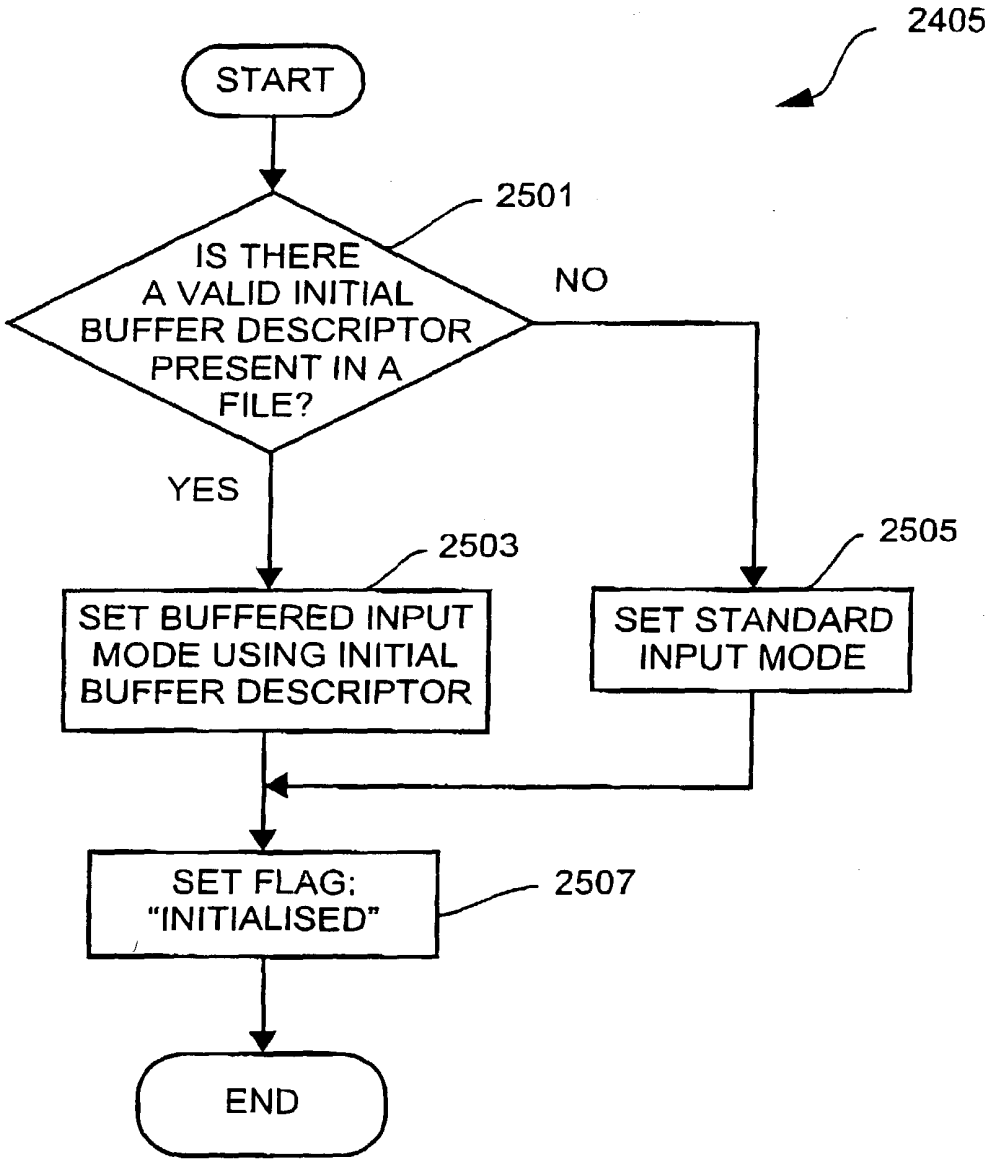

2405
START
2501
IS THERE A VALID INITIAL BUFFER DESCRIPTOR PRESENT IN A FILE?
NO
YES
2503
SET BUFFERED INPUT MODE USING INITIAL BUFFER DESCRIPTOR
2505
SET STANDARD INPUT MODE
SET FLAG: "INITIALISED"
2507
END

Fig. 25

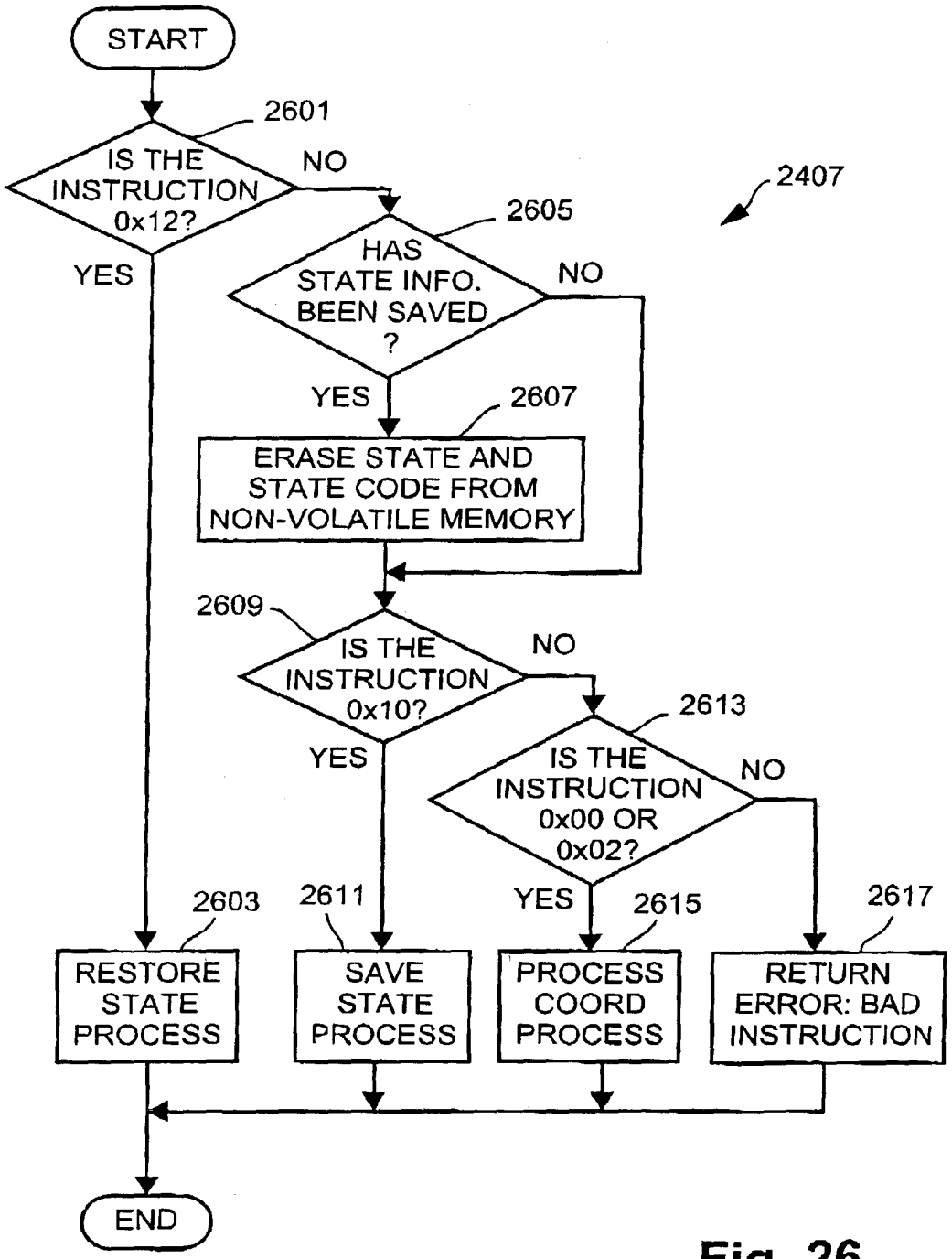

START
2601
IS THE INSTRUCTION 0x12?
NO
YES
2407
2605
HAS STATE INFO. BEEN SAVED ?
NO
YES
2607
ERASE STATE AND STATE CODE FROM NON-VOLATILE MEMORY
2609
IS THE INSTRUCTION 0x10?
NO
YES
2613
IS THE INSTRUCTION 0x00 OR 0x02?
NO
YES
2603
2611
2615
2617
RESTORE STATE PROCESS
SAVE STATE PROCESS
PROCESS COORD PROCESS
RETURN ERROR: BAD INSTRUCTION
END

Fig. 26

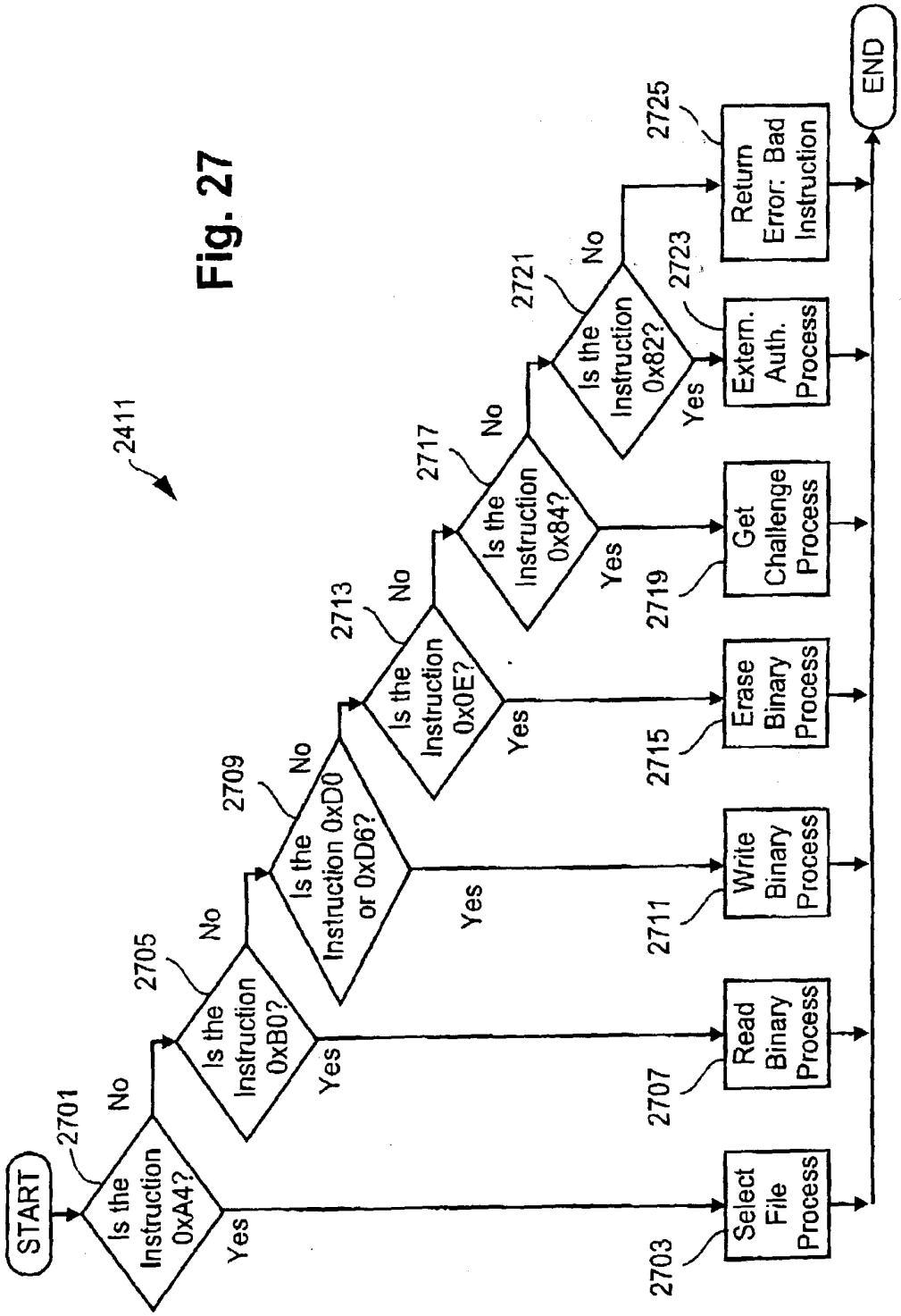
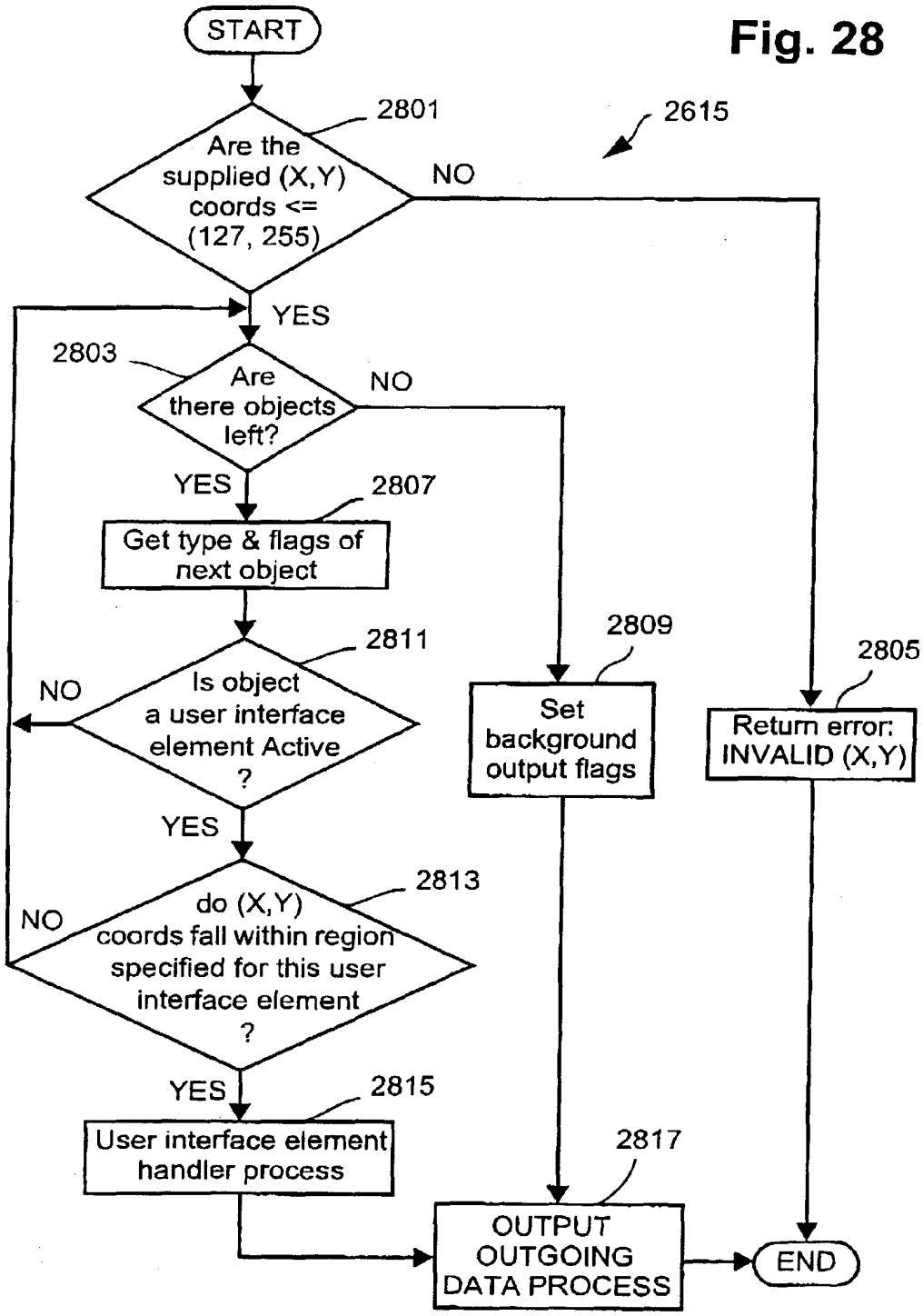
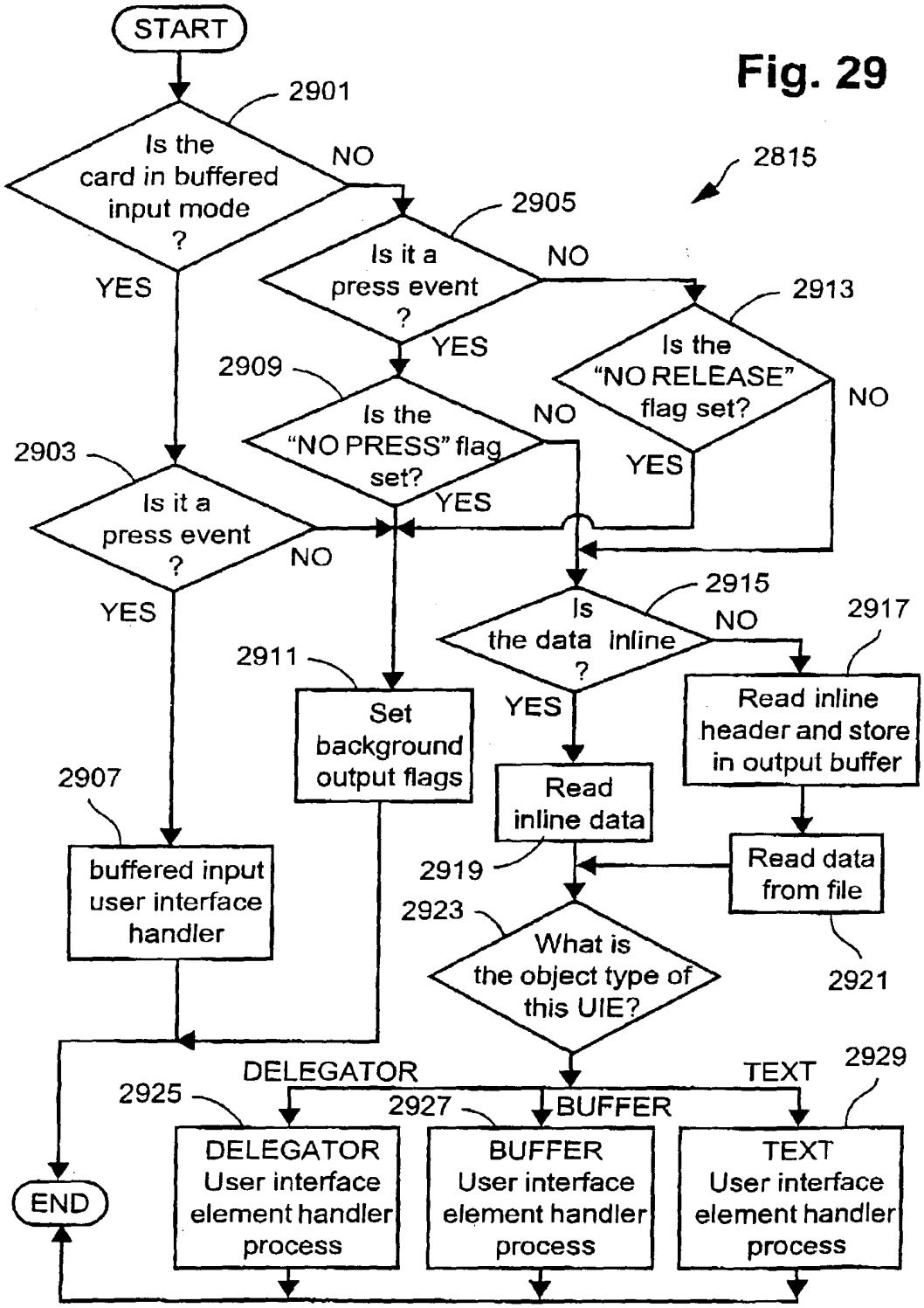

2411
START
2701
Is the Instruction 0xA4?
Yes
No
2703
Select File Process
2705
Is the Instruction 0xB0?
Yes
No
2707
Read Binary Process
2709
Is the Instruction 0xD0 or 0xD6?
Yes
No
2711
Write Binary Process
2713
Is the Instruction 0xE0?
Yes
No
2715
Erase Binary Process
2717
Is the Instruction 0x84?
Yes
No
2719
Get Challenge Process
2721
Is the Instruction 0x82?
Yes
No
2723
Extern. Auth. Process
2725
Return Error: Bad Instruction
END

Fig. 27


START
Fig. 28
2615
2801
Are the supplied (X,Y) coords <= (127, 255)
NO
YES
2803
Are there objects left?
NO
YES
2807
Get type & flags of next object
2811
Is object a user interface element Active ?
NO
YES
2813
do (X,Y) coords fall within region specified for this user interface element ?
NO
YES
2815
User interface element handler process
2809
Set background output flags
2805
Return error: INVALID (X,Y)
2817
OUTPUT OUTGOING DATA PROCESS
END


START
Fig. 29
2815
2901
Is the card in buffered input mode ?
NO
YES
2905
Is it a press event ?
NO
YES
2913
Is the "NO RELEASE" flag set?
NO
YES
2909
Is the "NO PRESS" flag set?
NO
YES
2903
Is it a press event ?
NO
YES
2915
Is the data inline ?
NO
YES
2917
Read inline header and store in output buffer
2911
Set background output flags
2907
buffered input user interface handler
Read inline data
2919
Read data from file
2921
2923
What is the object type of this UIE?
DELEGATOR
BUFFER
TEXT
2925
2927
2929
DELEGATOR User interface element handler process
BUFFER User interface element handler process
TEXT User interface element handler process
END

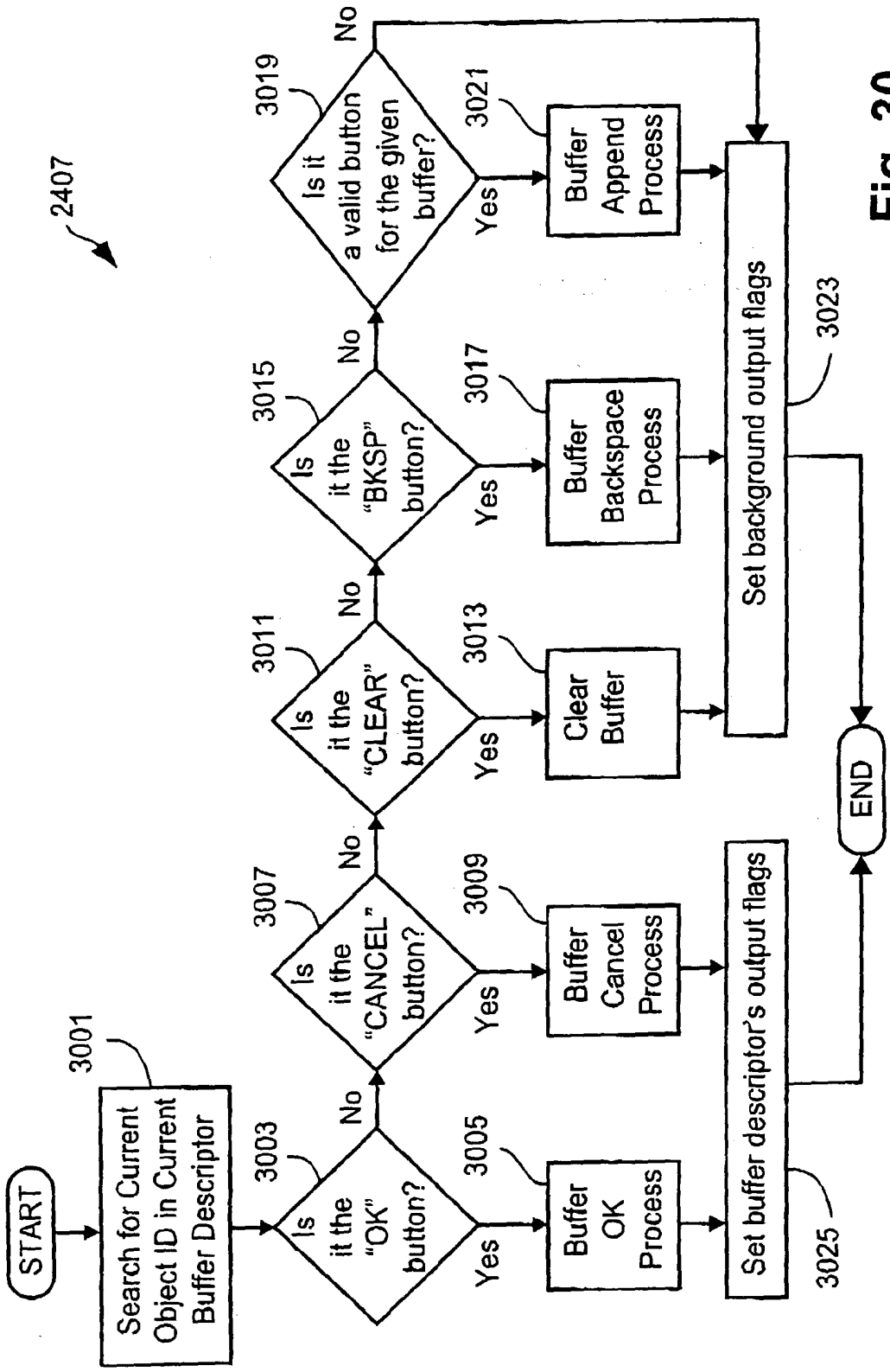

2407
START
3001
Search for Current Object ID in Current Buffer Descriptor
3003
Is it the "OK" button?
Yes
No
3005
Buffer OK Process
3007
Is it the "CANCEL" button?
Yes
No
3009
Buffer Cancel Process
3011
Is it the "CLEAR" button?
Yes
No
3013
Clear Buffer
3015
Is it the "BKSP" button?
Yes
No
3017
Buffer Backspace Process
3019
Is it a valid button for the given buffer?
Yes
No
3021
Buffer Append Process
Set buffer descriptor's output flags
3025
Set background output flags
3023
END

Fig. 30

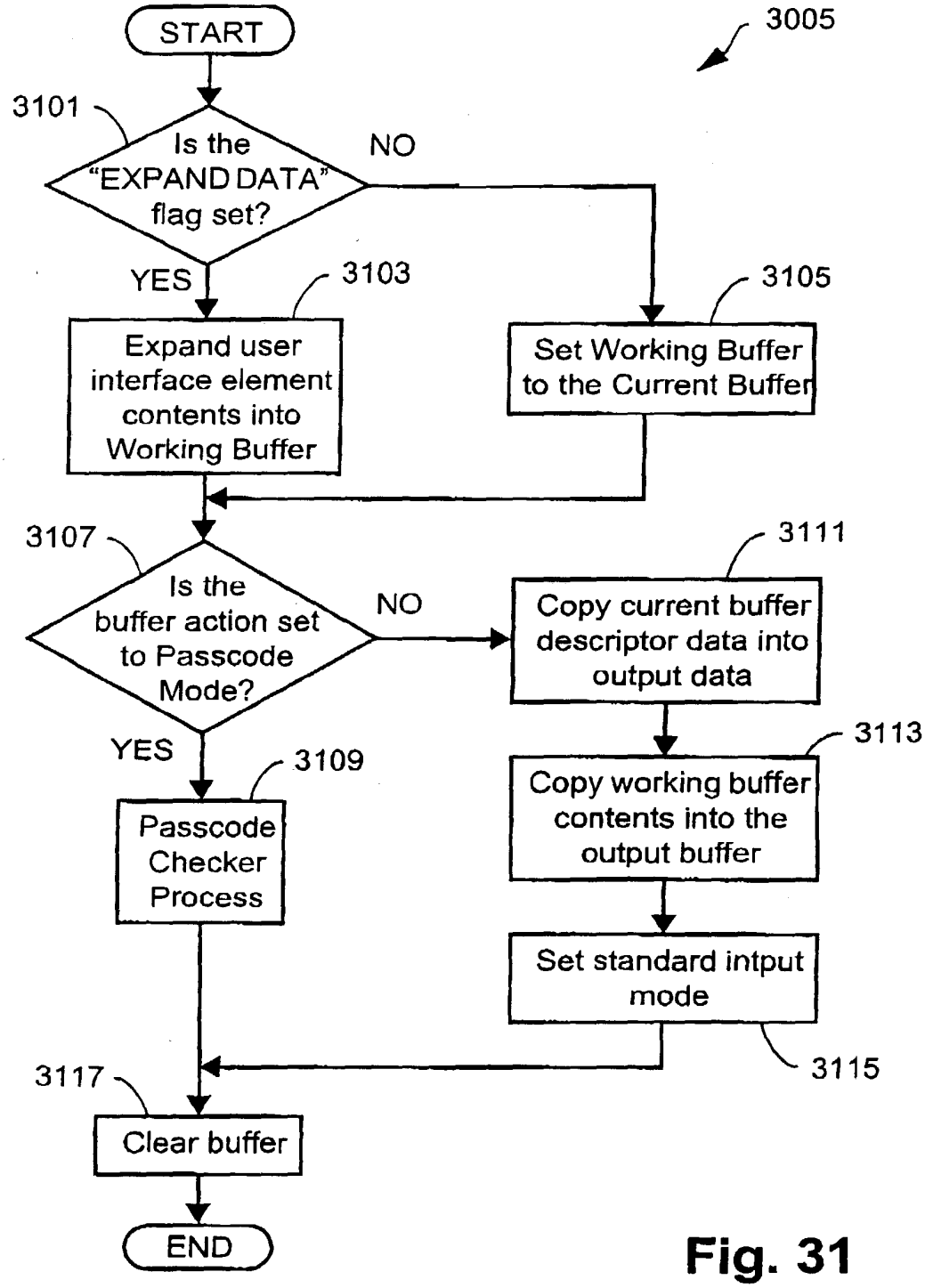

START
3005
3101
Is the "EXPAND DATA" flag set?
NO
YES
3103
Expand user interface element contents into Working Buffer
3105
Set Working Buffer to the Current Buffer
3107
Is the buffer action set to Passcode Mode?
NO
3111
Copy current buffer descriptor data into output data
YES
3109
Passcode Checker Process
3113
Copy working buffer contents into the output buffer
Set standard intput mode
3115
3117
Clear buffer
END

Fig. 31

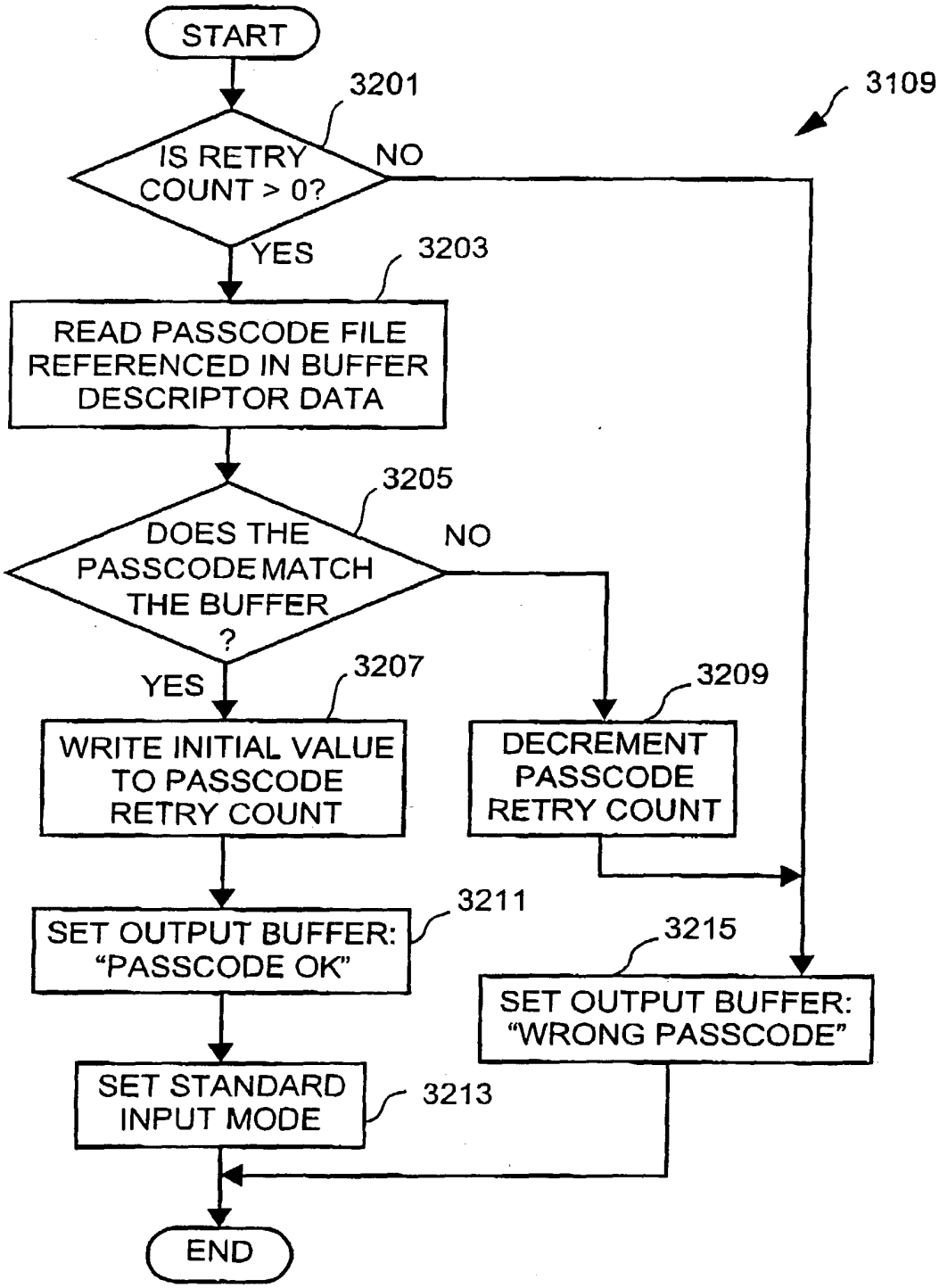

START
3201
3109
IS RETRY COUNT > 0?
NO
YES
3203
READ PASSCODE FILE REFERENCED IN BUFFER DESCRIPTOR DATA
3205
DOES THE PASSCODE MATCH THE BUFFER ?
NO
YES
3207
3209
WRITE INITIAL VALUE TO PASSCODE RETRY COUNT
DECREMENT PASSCODE RETRY COUNT
3211
SET OUTPUT BUFFER: "PASSCODE OK"
3215
SET OUTPUT BUFFER: "WRONG PASSCODE"
SET STANDARD INPUT MODE
3213
END

Fig. 32

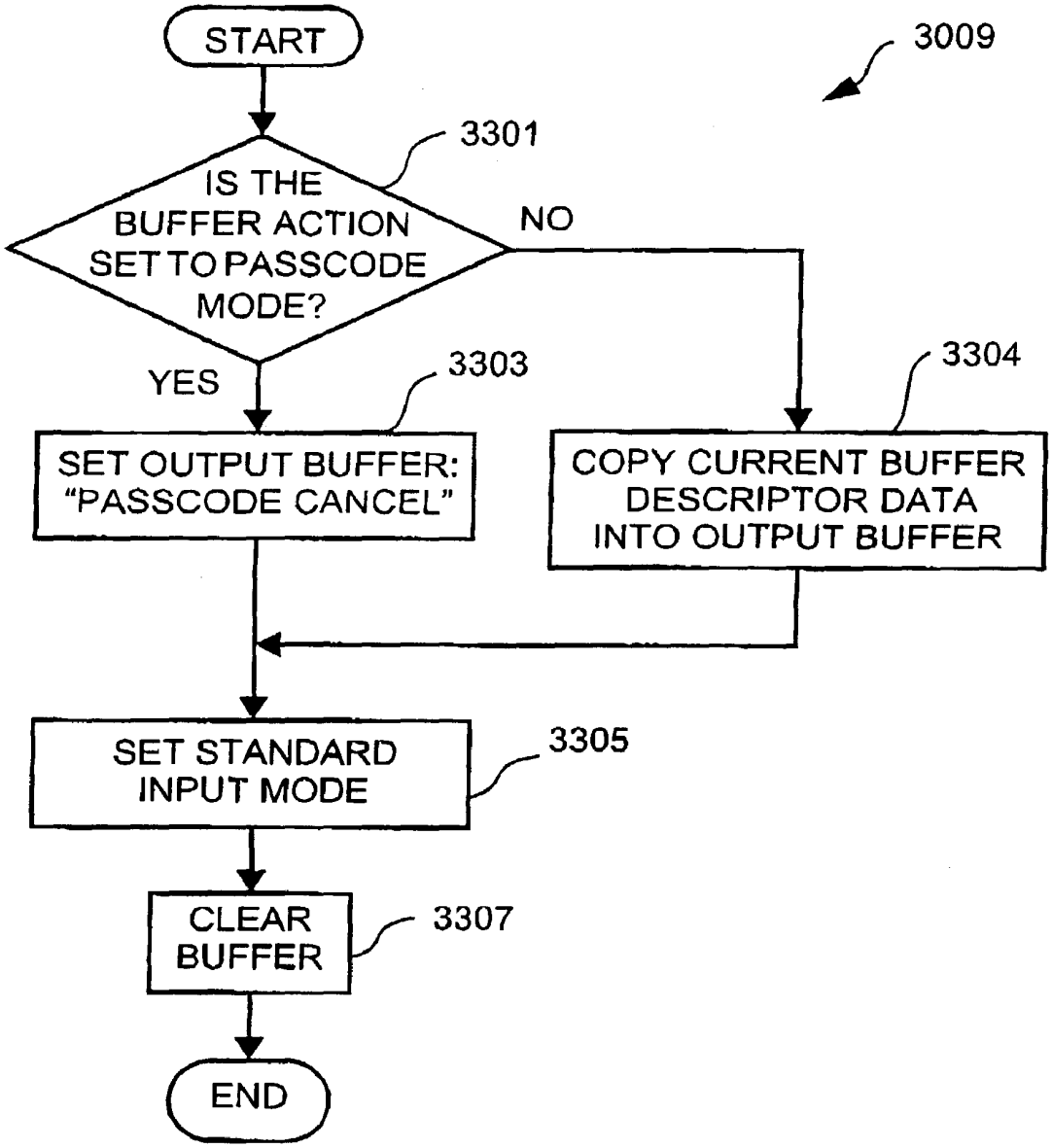

START
3009
3301
IS THE BUFFER ACTION SET TO PASSCODE MODE?
NO
YES
3303
3304
SET OUTPUT BUFFER: "PASSCODE CANCEL"
COPY CURRENT BUFFER DESCRIPTOR DATA INTO OUTPUT BUFFER
SET STANDARD INPUT MODE
3305
CLEAR BUFFER
3307
END

Fig. 33

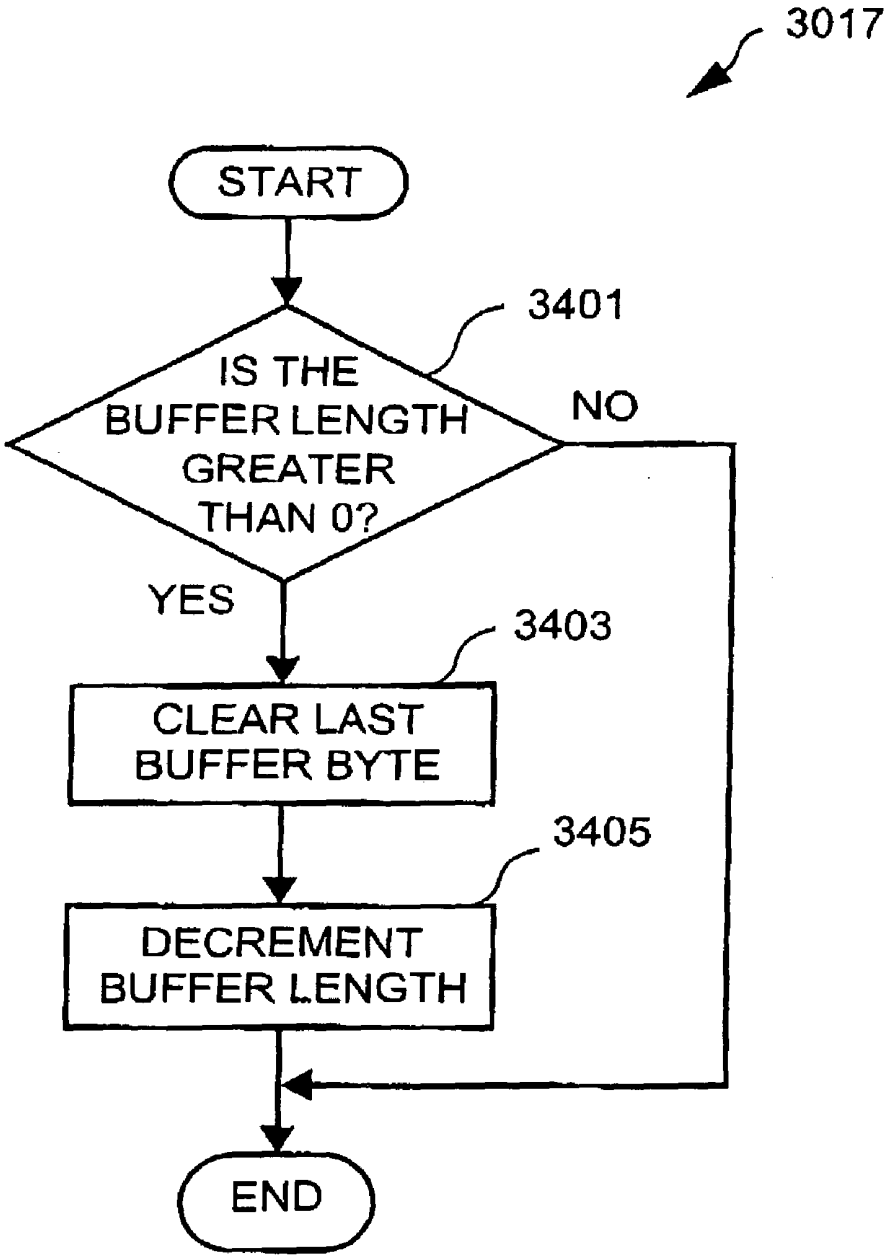

3017
START
3401
IS THE BUFFER LENGTH GREATER THAN 0?
NO
YES
3403
CLEAR LAST BUFFER BYTE
3405
DECREMENT BUFFER LENGTH
END

Fig. 34

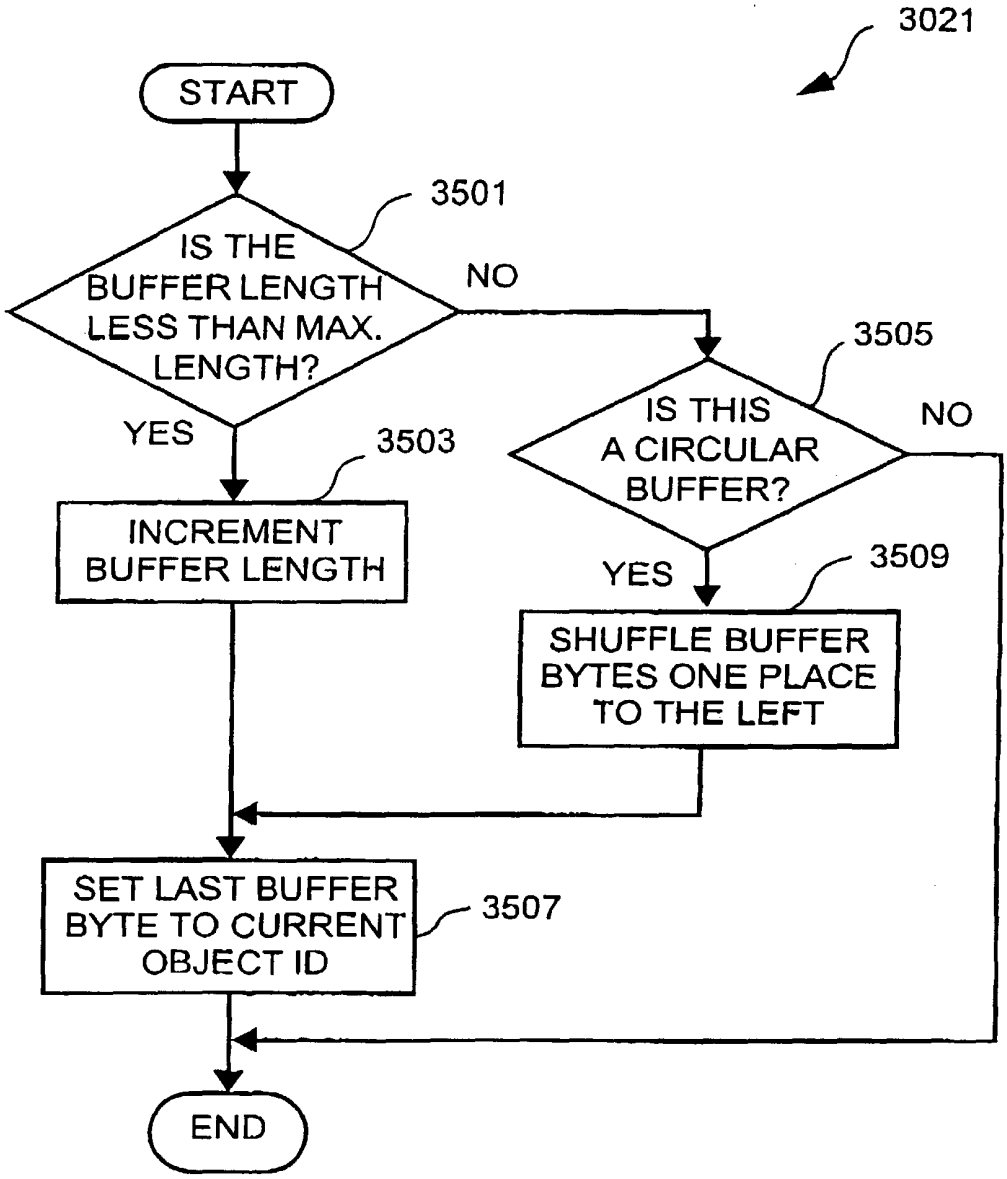

3021
START
3501
IS THE BUFFER LENGTH LESS THAN MAX. LENGTH?
NO
YES
3503
INCREMENT BUFFER LENGTH
3505
IS THIS A CIRCULAR BUFFER?
NO
YES
3509
SHUFFLE BUFFER BYTES ONE PLACE TO THE LEFT
SET LAST BUFFER BYTE TO CURRENT OBJECT ID
3507
END

Fig. 35

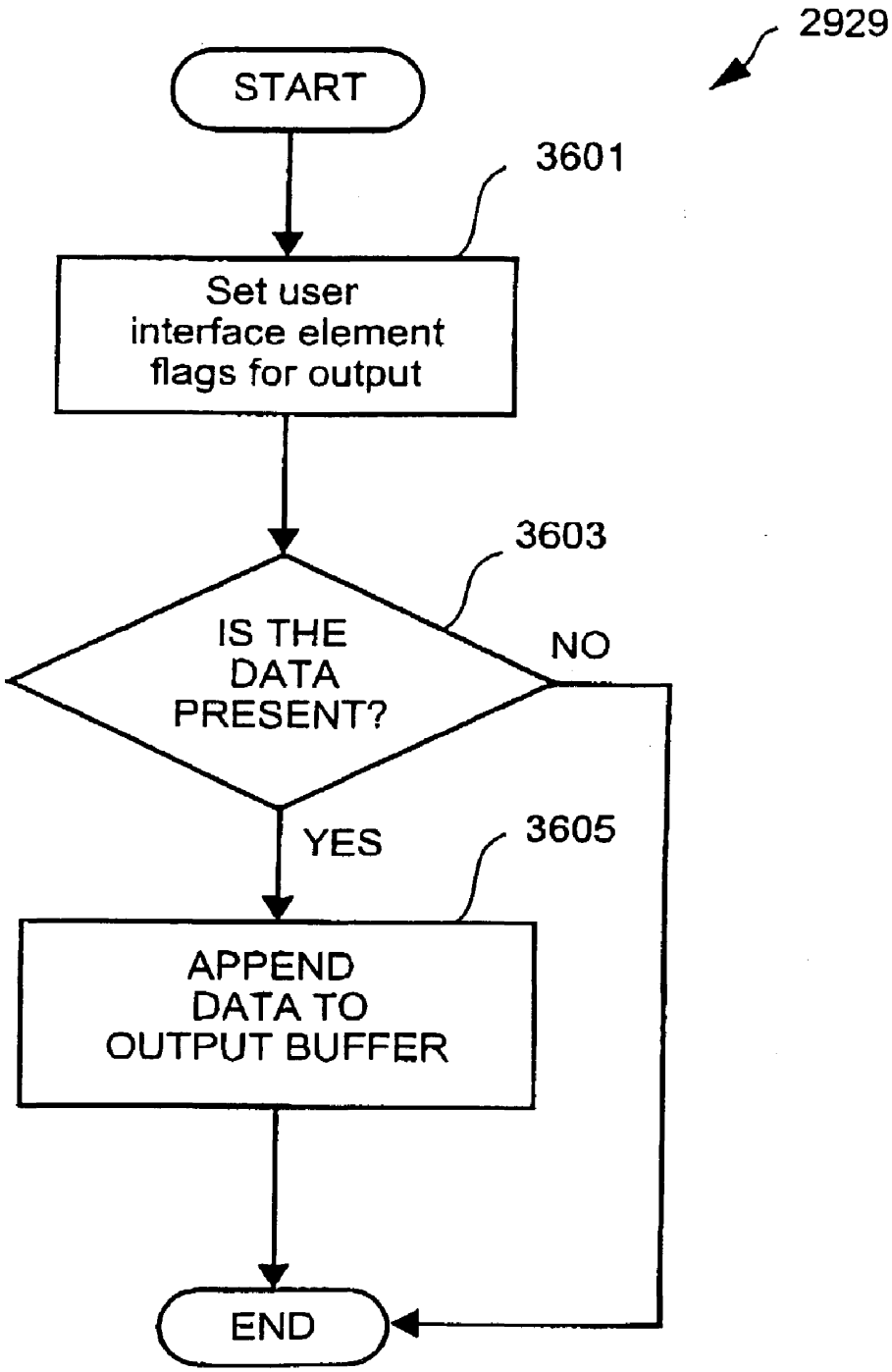

2929
START
3601
Set user interface element flags for output
3603
IS THE DATA PRESENT?
NO
YES
3605
APPEND DATA TO OUTPUT BUFFER
END

Fig. 36

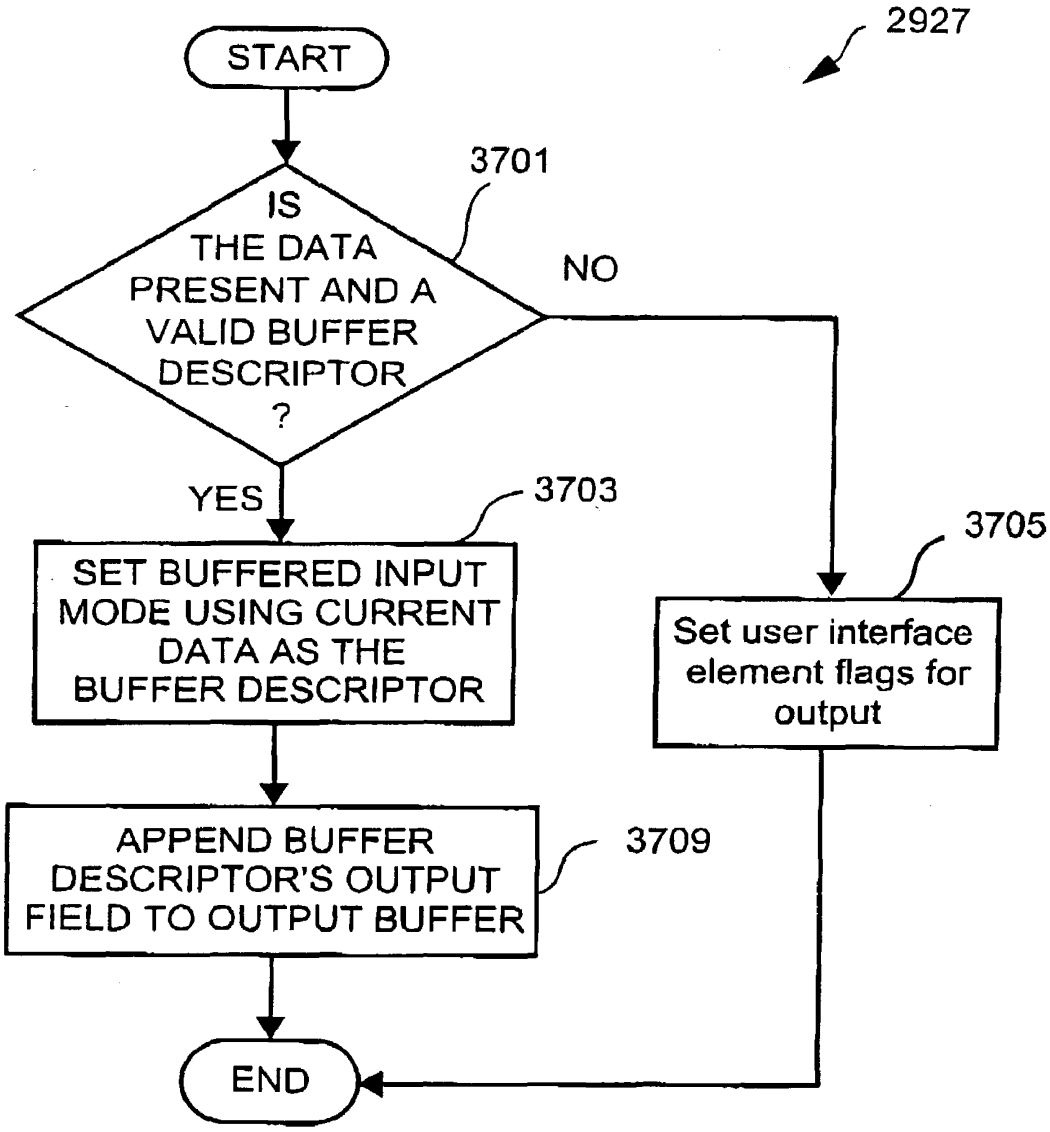

2927
START
3701
IS THE DATA PRESENT AND A VALID BUFFER DESCRIPTOR ?
NO
YES
3703
SET BUFFERED INPUT MODE USING CURRENT DATA AS THE BUFFER DESCRIPTOR
3705
Set user interface element flags for output
APPEND BUFFER DESCRIPTOR'S OUTPUT FIELD TO OUTPUT BUFFER
3709
END

Fig. 37

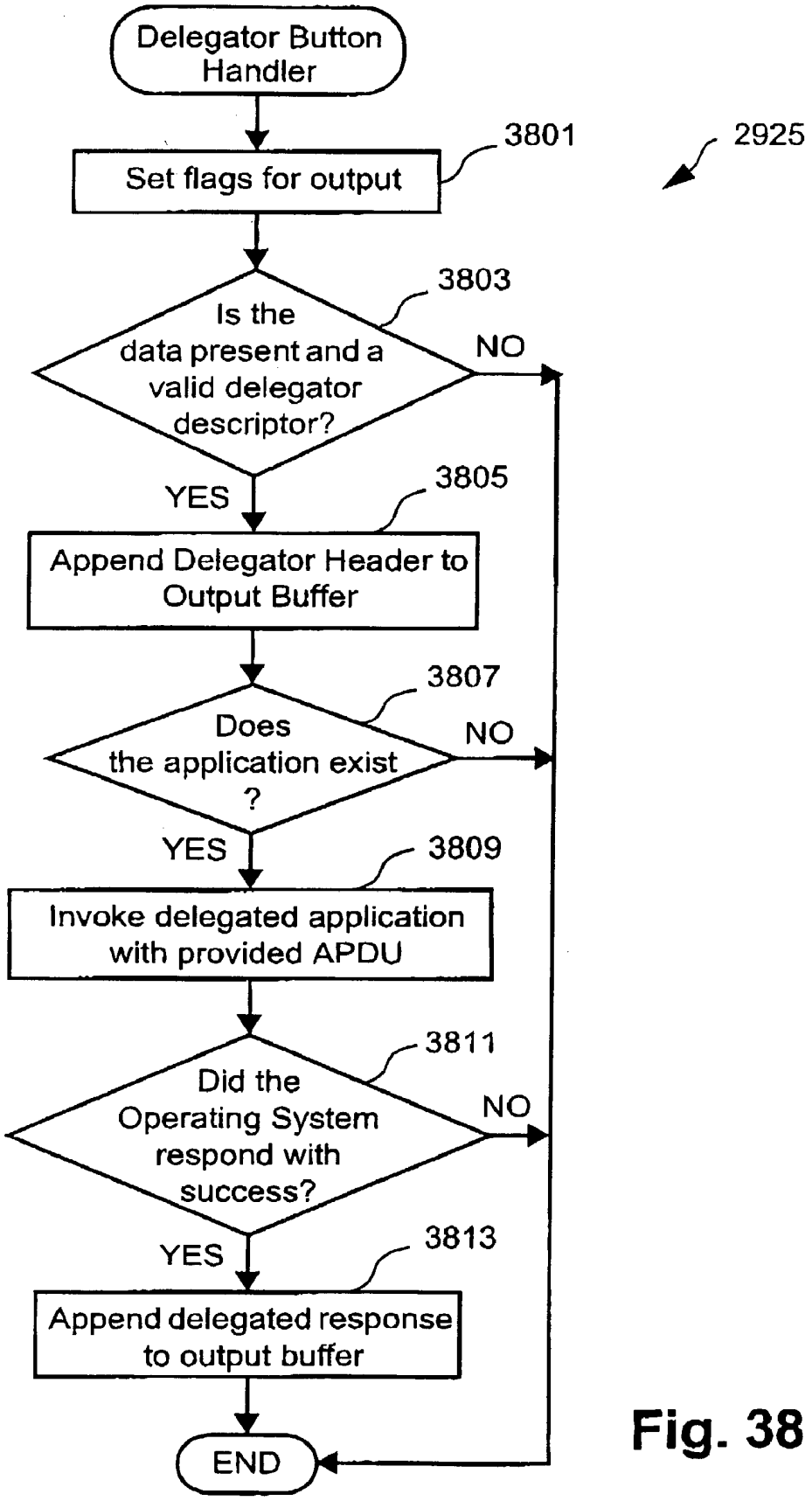

Delegator Button Handler
Set flags for output
3801
2925
Is the data present and a valid delegator descriptor?
3803
NO
YES
3805
Append Delegator Header to Output Buffer
Does the application exist ?
3807
NO
YES
3809
Invoke delegated application with provided APDU
Did the Operating System respond with success?
3811
NO
YES
3813
Append delegated response to output buffer
END

Fig. 38

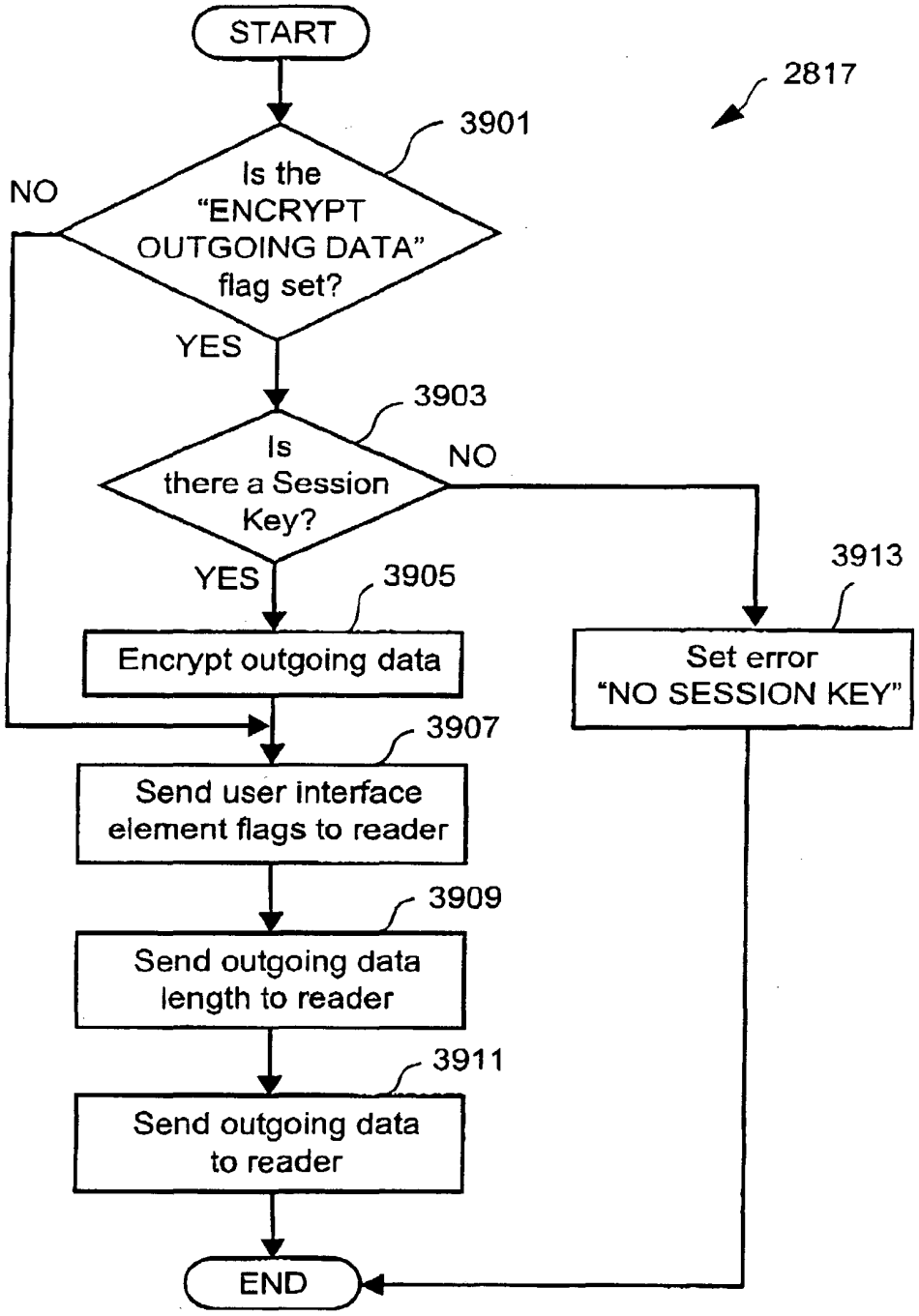

START
2817
3901
Is the "ENCRYPT OUTGOING DATA" flag set?
NO
YES
3903
Is there a Session Key?
NO
YES
3905
Encrypt outgoing data
3913
Set error "NO SESSION KEY"
3907
Send user interface element flags to reader
3909
Send outgoing data length to reader
3911
Send outgoing data to reader
END

Fig. 39

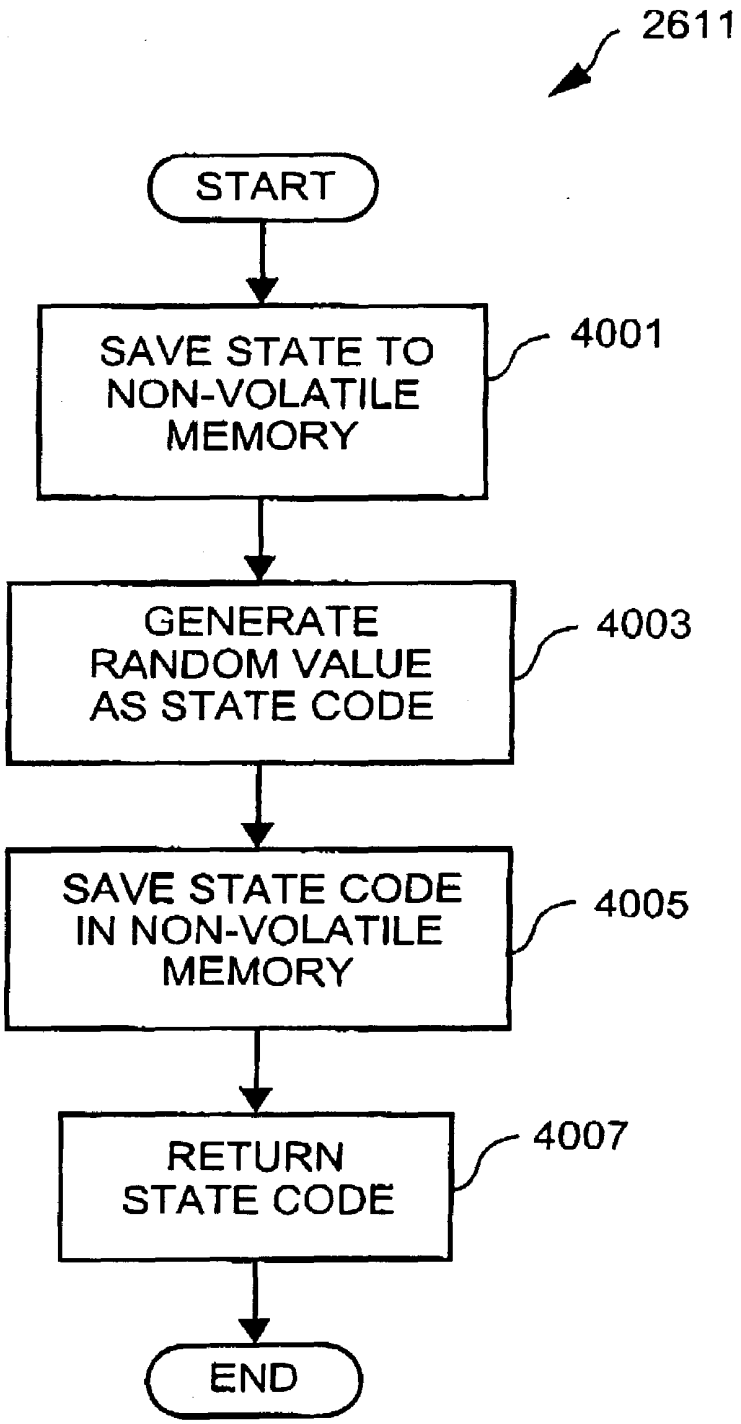

2611
START
SAVE STATE TO NON-VOLATILE MEMORY
4001
GENERATE RANDOM VALUE AS STATE CODE
4003
SAVE STATE CODE IN NON-VOLATILE MEMORY
4005
RETURN STATE CODE
4007
END

Fig. 40

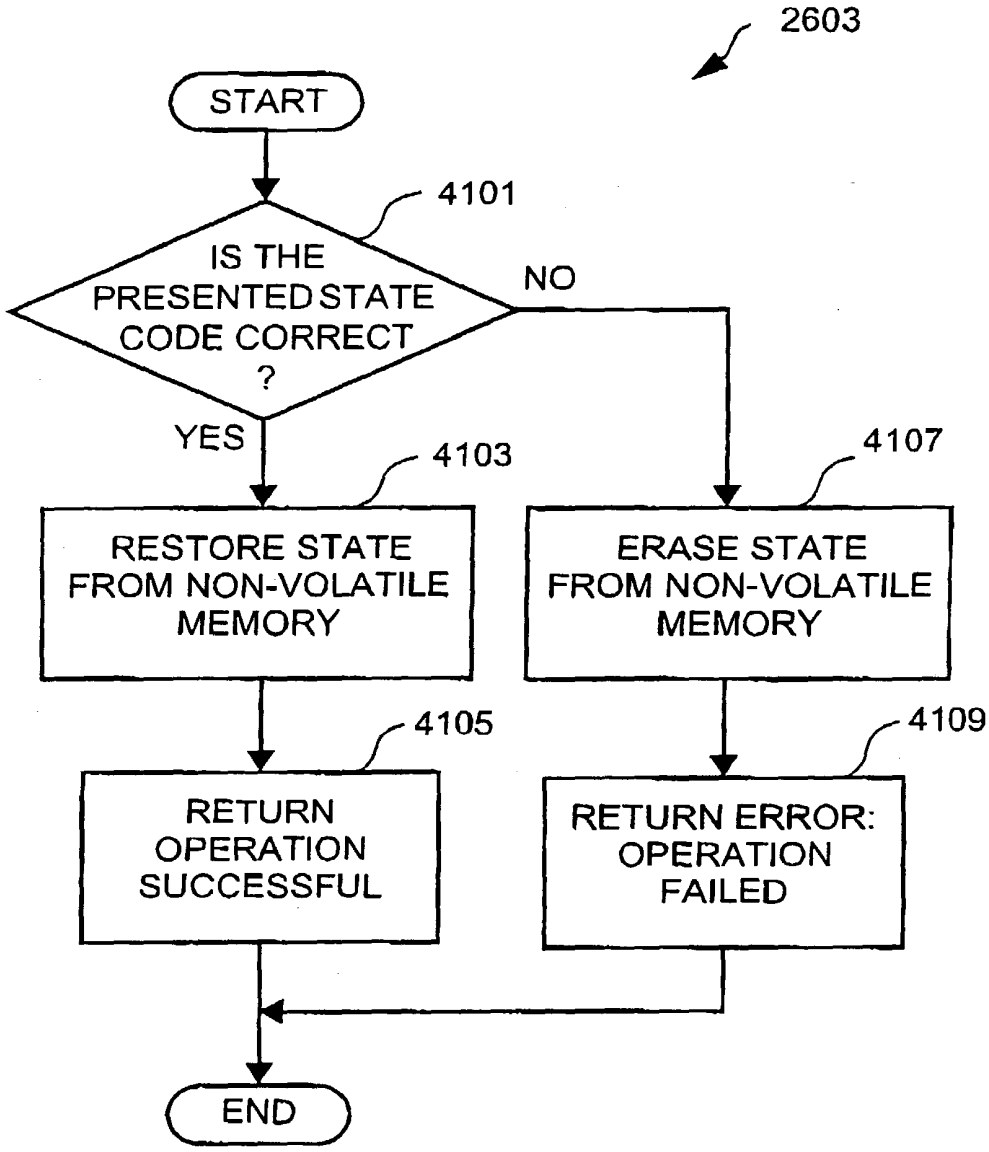

2603
START
4101
IS THE PRESENTED STATE CODE CORRECT ?
NO
YES
4103
RESTORE STATE FROM NON-VOLATILE MEMORY
4107
ERASE STATE FROM NON-VOLATILE MEMORY
4105
RETURN OPERATION SUCCESSFUL
4109
RETURN ERROR: OPERATION FAILED
END

Fig. 41

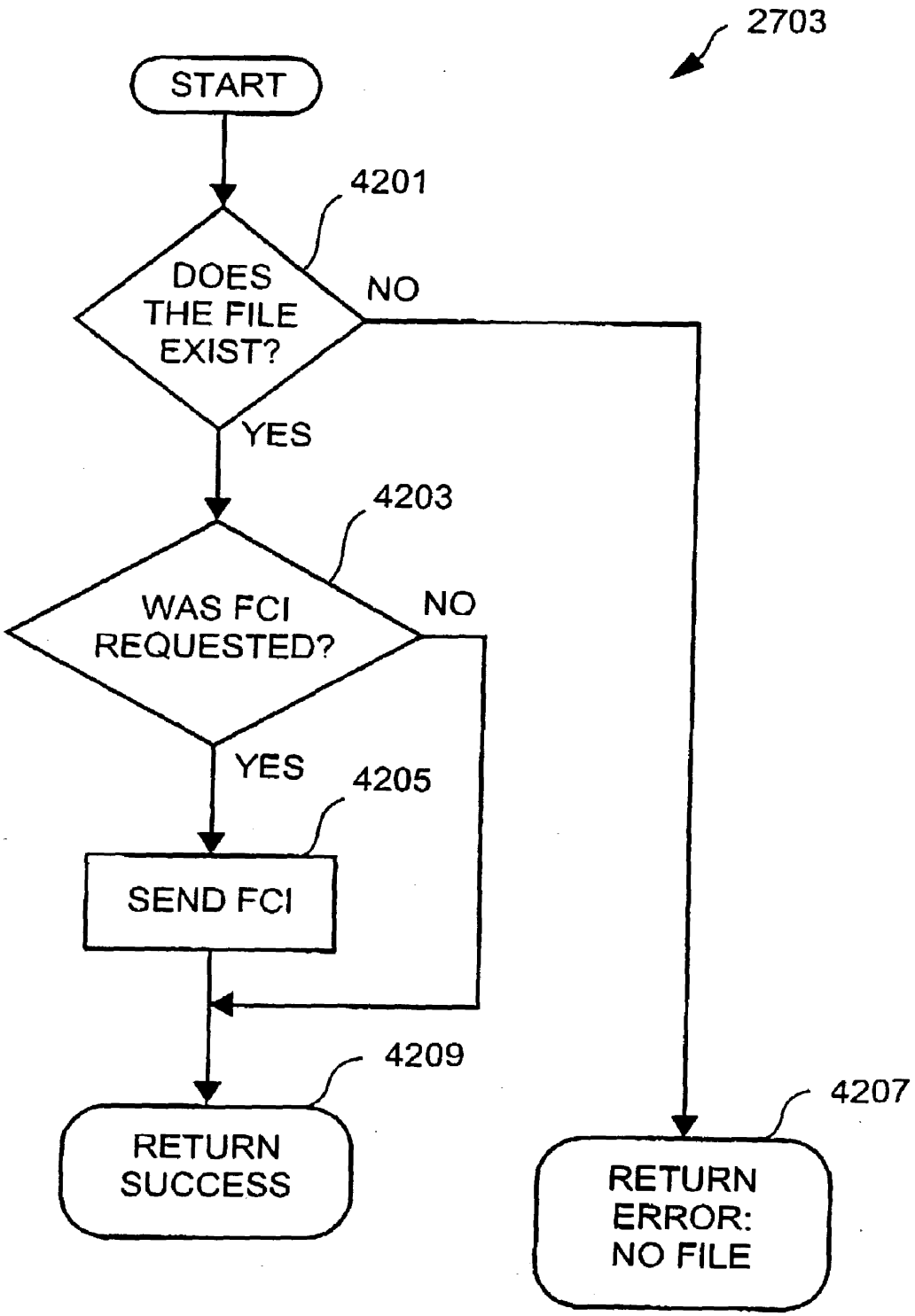

2703
START
4201
DOES THE FILE EXIST?
NO
YES
4203
WAS FCI REQUESTED?
NO
YES
4205
SEND FCI
4209
RETURN SUCCESS
4207
RETURN ERROR: NO FILE

Fig. 42

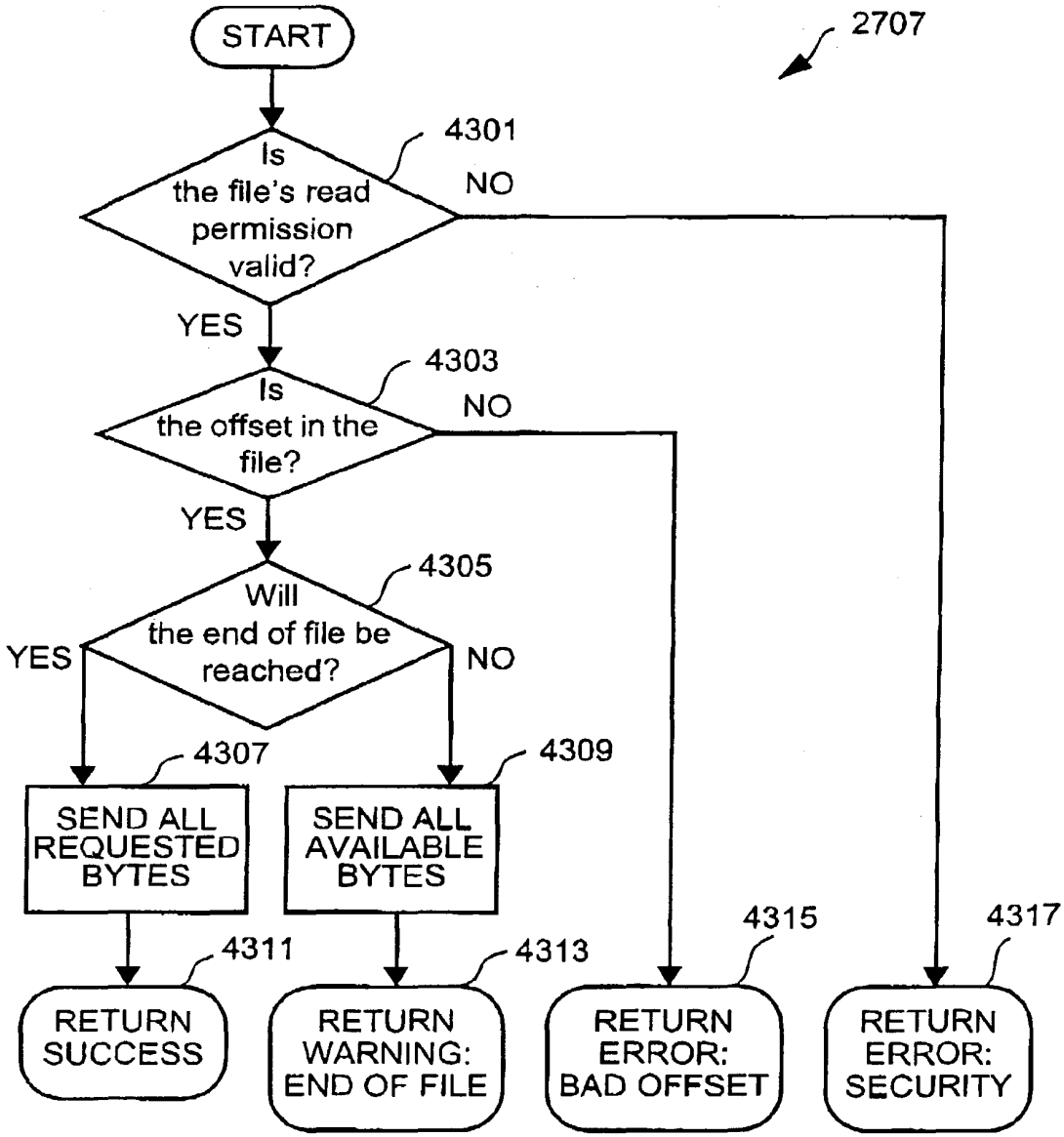

START
2707
4301
Is the file's read permission valid?
NO
YES
4303
Is the offset in the file?
NO
YES
4305
Will the end of file be reached?
YES
NO
4307
SEND ALL REQUESTED BYTES
4309
SEND ALL AVAILABLE BYTES
4311
RETURN SUCCESS
4313
RETURN WARNING: END OF FILE
4315
RETURN ERROR: BAD OFFSET
4317
RETURN ERROR: SECURITY

Fig. 43

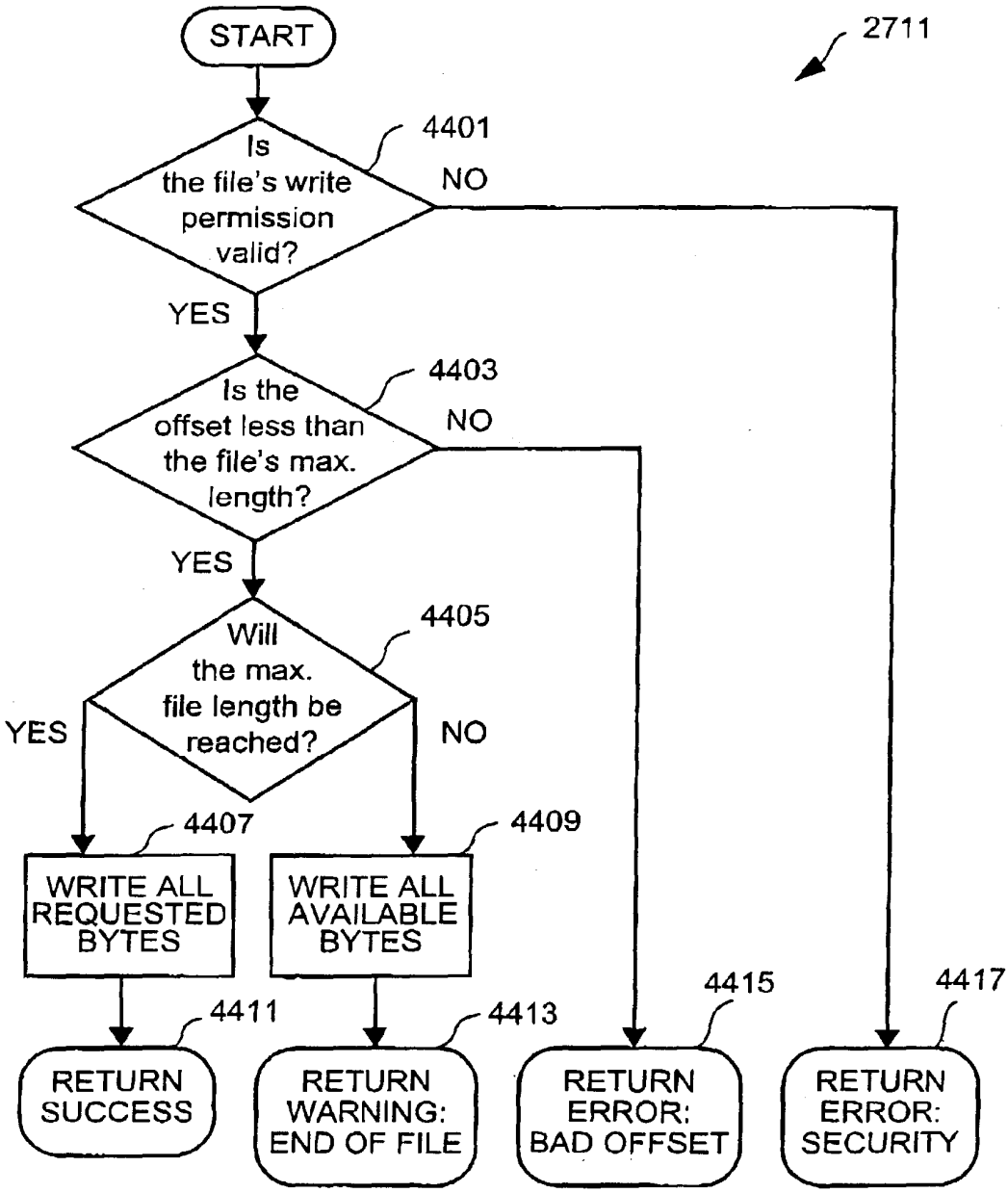

START
2711
Is the file's write permission valid?
4401
NO
YES
Is the offset less than the file's max. length?
4403
NO
YES
Will the max. file length be reached?
4405
YES
NO
4407
WRITE ALL REQUESTED BYTES
4409
WRITE ALL AVAILABLE BYTES
4411
RETURN SUCCESS
4413
RETURN WARNING: END OF FILE
4415
RETURN ERROR: BAD OFFSET
4417
RETURN ERROR: SECURITY

Fig. 44

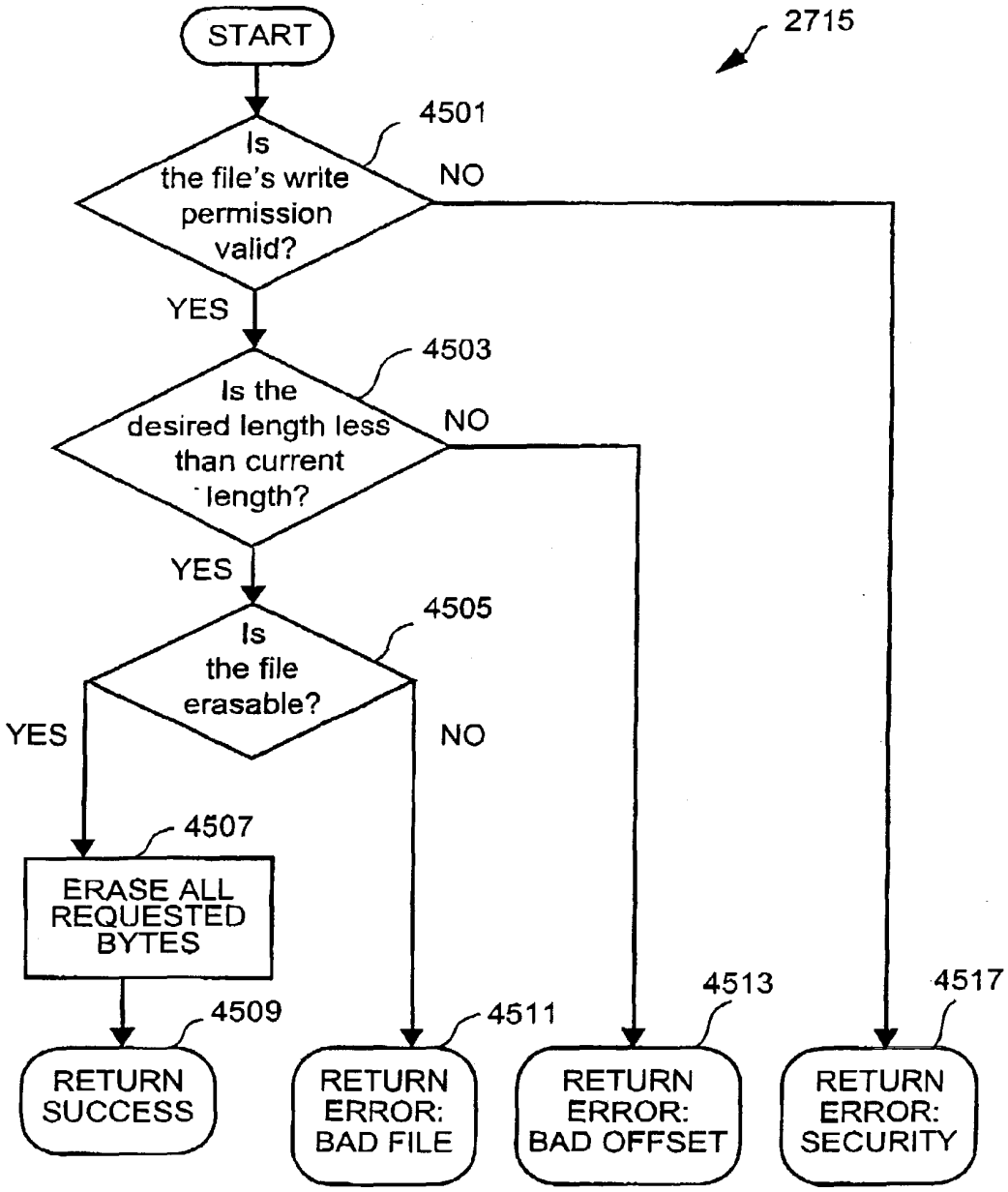

2715
START
4501
Is the file's write permission valid?
NO
YES
4503
Is the desired length less than current length?
NO
YES
4505
Is the file erasable?
YES
NO
4507
ERASE ALL REQUESTED BYTES
4509
RETURN SUCCESS
4511
RETURN ERROR: BAD FILE
4513
RETURN ERROR: BAD OFFSET
4517
RETURN ERROR: SECURITY

Fig. 45

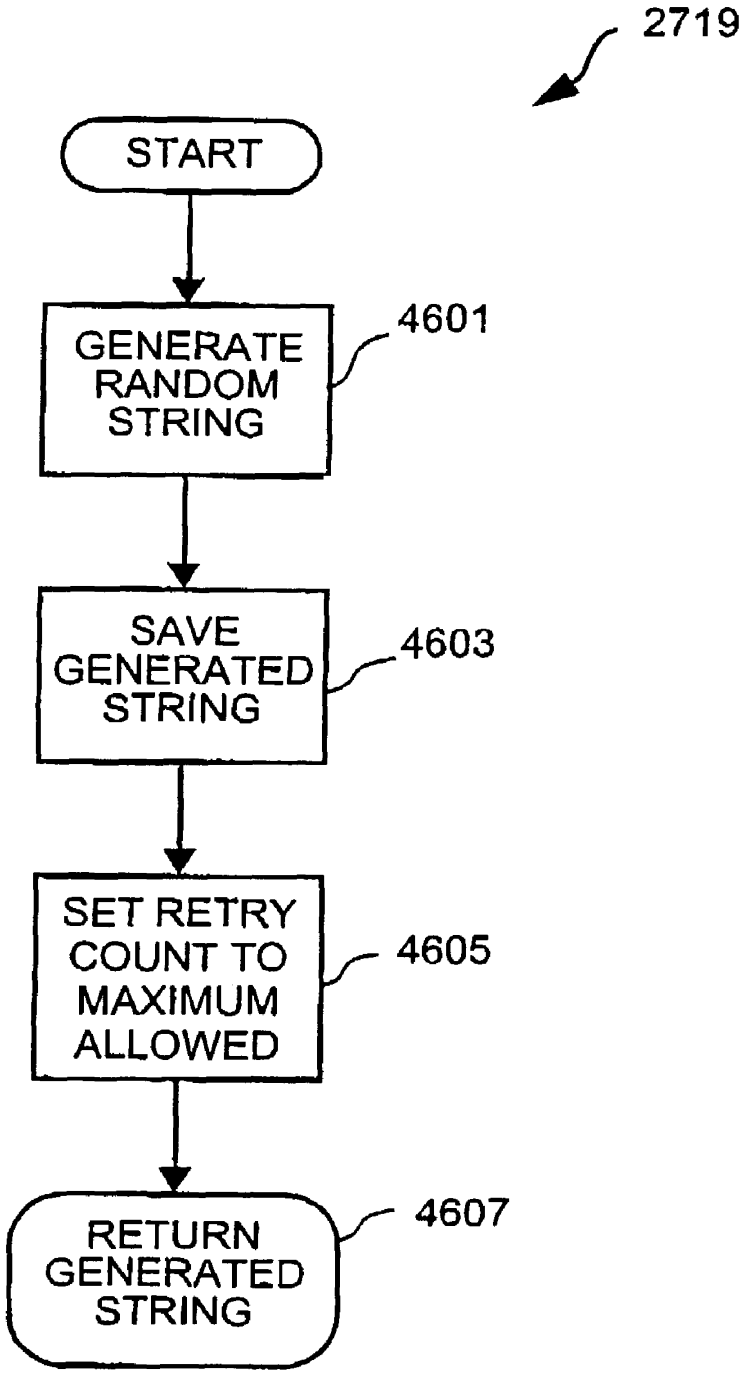

2719
START
GENERATE RANDOM STRING
4601
SAVE GENERATED STRING
4603
SET RETRY COUNT TO MAXIMUM ALLOWED
4605
RETURN GENERATED STRING
4607

Fig. 46

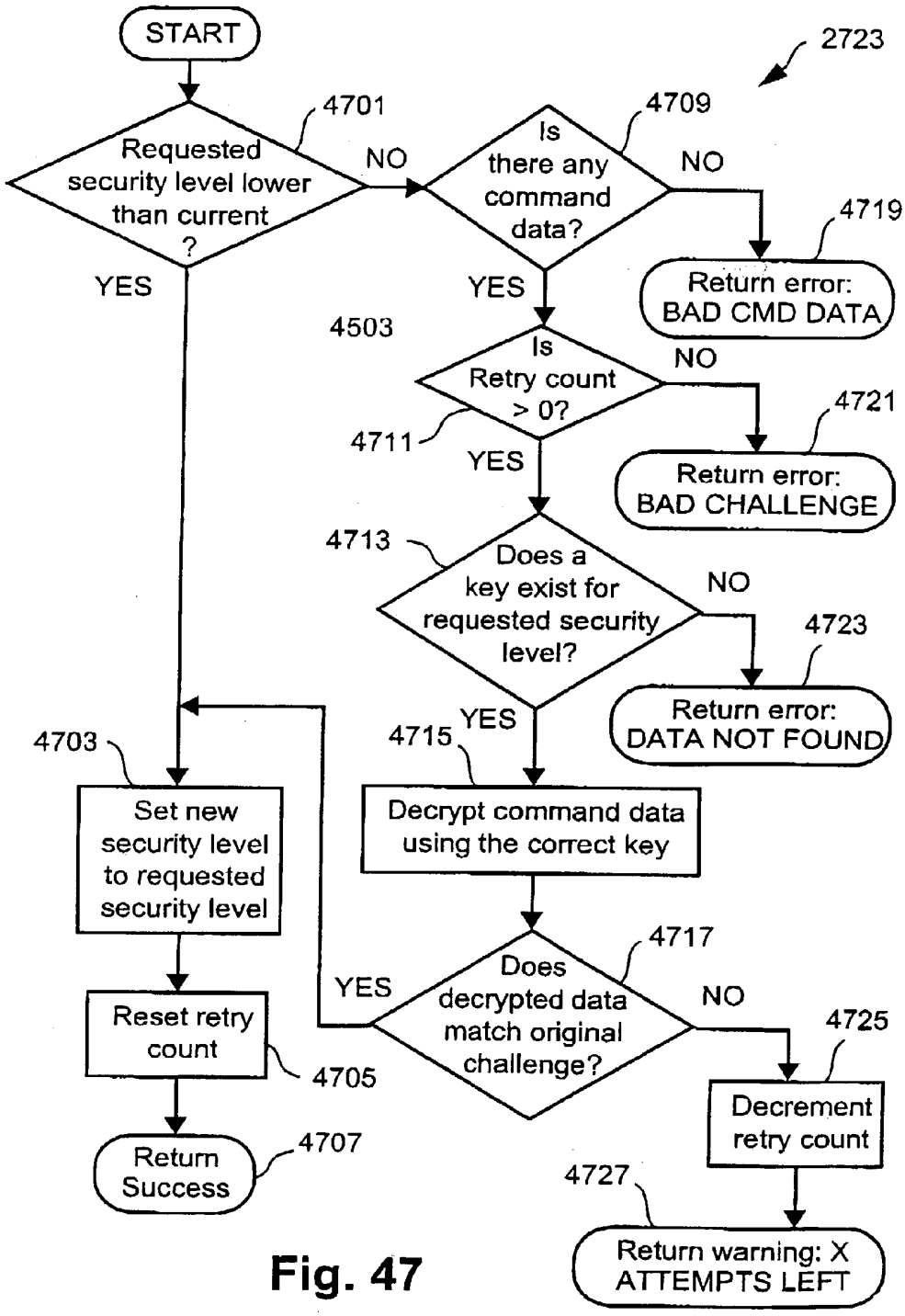

START
2723
4701
Requested security level lower than current ?
NO
YES
4709
Is there any command data?
NO
YES
4719
Return error: BAD CMD DATA
4503
Is Retry count > 0?
NO
4711
YES
4721
Return error: BAD CHALLENGE
4713
Does a key exist for requested security level?
NO
4723
Return error: DATA NOT FOUND
YES
4715
Decrypt command data using the correct key
4703
Set new security level to requested security level
4717
Does decrypted data match original challenge?
YES
NO
Reset retry count
4705
4725
Decrement retry count
Return Success
4707
4727
Return warning: X ATTEMPTS LEFT

Fig. 47

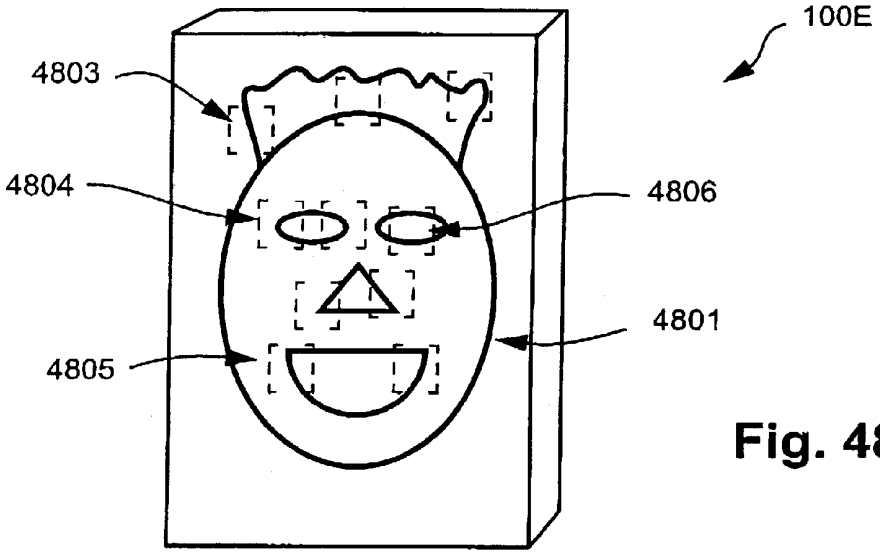


Fig. 48

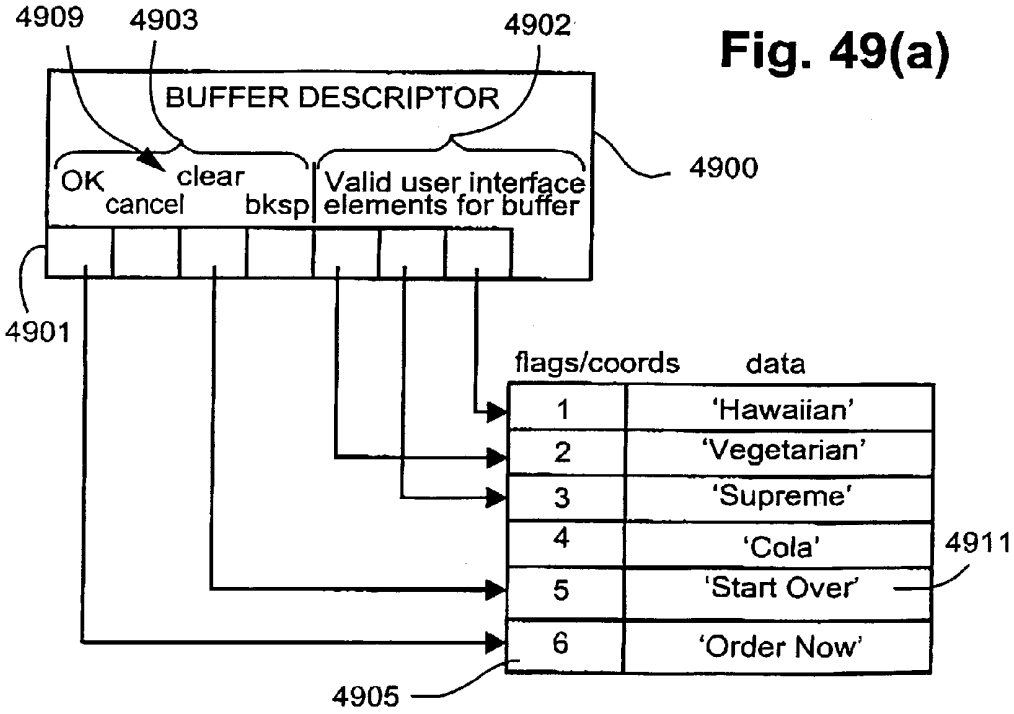


Fig. 49(a)

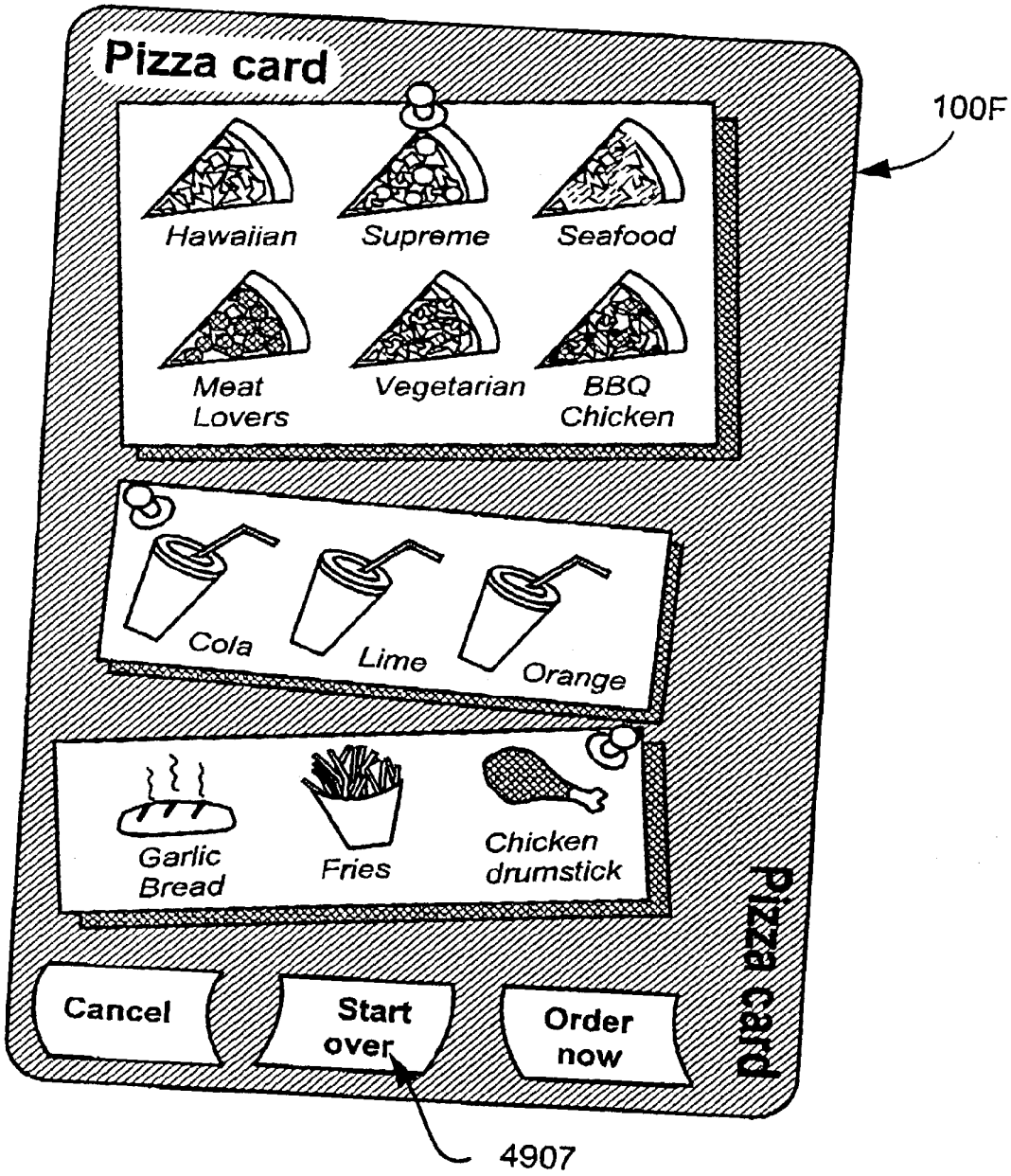

Pizza card
100F
Hawaiian
Supreme
Seafood
Meat Lovers
Vegetarian
BBQ Chicken
Cola
Lime
Orange
Garlic Bread
Fries
Chicken drumstick
Cancel
Start over
Order now
pizza card
4907

Fig. 49(b)

USER INTERFACE FOR ACCESSING FILES IN A SMARTCARD FILE

TECHNICAL FIELD OF THE INVENTION

[0001] The present invention relates generally to a control template or smart card for use with a reader device and, in particular, to a user interface for a microprocessor or central processing unit (CPU) smart card allowing a user to access a file stored on the CPU card. The invention also relates to a computer program product including a computer readable medium having recorded thereon a computer program providing a user interface for a microprocessor or central processing unit (CPU) smart card allowing a user to access a file stored on the CPU card.

BACKGROUND ART

[0002] Control cards, such as smart cards, often include some form of readable storage means such as a magnetic strip, an optical code (e.g. a bar code) or an on-board memory chip, for storing data (e.g. a personal identification number) associated with the card. Such control cards can be generically referred to as memory cards. However, control cards including a storage means in the form of an on-board memory chip are generally referred to as 'smart cards'. The data stored in the storage means is generally read by some form of terminal device, which includes a magnetic read head, for example.

[0003] Some smart cards include a microprocessor integrally formed within the smart card. These smart cards are generally referred to as microprocessor or central processing unit (CPU) cards.

[0004] One exisiting CPU card includes a user interface in the form of user interface elements printed on at least one surface of the card and a data structure describing the interface, where the data structure is stored in a memory integrally formed within the CPU card. A reader device used with the CPU cardincludes a transparent touch screen positioned above the CPU card so that user interface elements printed on a surface of the smart card are visible underneath the transparent touch screen. The reader device is configured to determine the position of a touch on the transparent screen and use the data structure information stored within a memory of the card to determine which user interface elements have been pressed. Data associated with the pressed user interface element is then returned to the reader which then sends a data string associated with the selected user interface elements to a remote application.

[0005] Typically CPU cards restrict a user interface definition associated with a user interface printed on a surface of the card from being modified. Therefore, data associated with individual user interface elements cannot be customised by a user or manufacturer of the card. For example, if the user of a card were to change his or her address, the user would need to be issued a new card which has been updated to include the details of the new address of the user or the card would need to be reprogrammed to include the details of the new address.

[0006] The "International Standards Organisation (ISO)/International Electrotechnical Commission (IEC) 7816-4 Standards" describe a file system designed for smart cards, together with commands that can be used to interface with the file system. However, this standard software interface is inaccessible to a user of a CPU card so that when a CPU card is used for a particular application, an associated terminal device must be programmed in order for correct commands to be issued to the CPU card in response to user input. If the same CPU card is used in a different smart card system, the second system is generally unable to generate the appropriate commands from the same user input due to lack of knowledge about the specific file system resident on the CPU card. The above mentioned standard does not allow a portion of data in a file to be write protected while allowing any remaining data in the file to be altered. Therefore, a file of such a system can either only contain wholly secured data or wholly unsecured data.

SUMMARY OF THE INVENTION

[0007] It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements.

[0008] According to one aspect of the present invention there is provided a program for allowing access to data associated with at least one file stored within an electronic card having one or more associated user interface elements, said electronic card comprising a card portion having an electronic apparatus attached thereto, said electronic apparatus comprising a processor means coupled to a memory means, said processor means being adapted for coupling to a reading device to facilitate reading of said electronic card, said memory means being configured to retain said files and said program for execution by said processor means, said program comprising:

[0009] code for relating reading signals generated from a user selection of at least one of said associated user interface elements with a first file stored within said memory, said first file being classified by said program as protected; and

[0010] code for allowing access to data of a second file associated with said first file, said second file being classified by said program as unprotected.

[0011] According to one aspect of the present invention there is provided a method of allowing access to data associated with at least one file stored within an electronic card having one or more associated user interface elements, said electronic card comprising a card portion having an electronic apparatus attached thereto, said electronic apparatus comprising a processor means coupled to a memory means, said processor means being adapted for coupling to a reading device to facilitate reading of said electronic card, said memory means being configured to retain at least said files, said method comprising the steps of:

[0012] relating reading signals generated from a user selection of at least one of said associated user interface elements with a first file stored within said memory, said first file being classified as protected; and

[0013] allowing access to data of a second file associated with said first file, said second file being classified as unprotected.

[0014] According to one aspect of the present invention there is provided an apparatus for allowing access to data

associated with at least one file stored within an electronic card having one or more associated user interface elements, said electronic card comprising a card portion having an electronic apparatus attached thereto, said electronic apparatus comprising a processor means coupled to a memory means, said processor means being adapted for coupling to a reading device to facilitate reading of said electronic card, said memory means being configured to retain at least said files, said apparatus comprising:

[0015] relating means for relating reading signals generated from a user selection of at least one of said associated user interface elements with a first file stored within said memory, said first file being classified as protected; and

[0016] access allowing means for allowing access to data of a second file associated with said first file, said second file being classified as unprotected.

[0017] Other aspects of the invention are also disclosed.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] One or more embodiments of the present invention will now be described with reference to the drawings, in which:

[0019] **FIG. 1**(*a*) is a perspective view of a smart card;

[0020] **FIG. 1**(*b*) is a perspective view of another smart card;

[0021] **FIG. 2**(*a*) is a longitudinal cross-sectional view taken along the line II(*a*)-II(*a*) of the card shown in **FIG. 1**(*a*);

[0022] **FIG. 2**(*b*) is a longitudinal cross-sectional view taken along the line II(*b*)-II(*b*);

[0023] **FIG. 3** is a perspective view of a reader device configured for use with the smart cards of FIGS. **1**(*a*) and **1**(*b*) of the card shown in **FIG. 1**(*b*);

[0024] **FIG. 4** shows a user inserting a smart card into the reader of **FIG. 3**;

[0025] **FIG. 5** shows a user operating the reader of **FIG. 3** after the smart card has been fully inserted;

[0026] **FIG. 6**(*a*) shows a hardware architecture for a smart card interface system;

[0027] **FIG. 6**(*b*) shows another hardware architecture for a smart card interface system;

[0028] **FIG. 7** is a schematic block diagram of the general-purpose computer of FIGS. **6**(*a*) and **6**(*b*) in more detail;

[0029] **FIG. 8** is schematic block diagram showing the set top box of **FIG. 6**(*b*) in more detail;

[0030] **FIG. 9** is a schematic block diagram representation of a smart card interface system software architecture;

[0031] **FIG. 10** is a schematic block diagram showing the internal configuration of the reader of **FIG. 3**;

[0032] **FIG. 11** shows the data structure of a card header as stored in the smart cards of **FIG. 1**(*a*) and **FIG. 1**(*b*);

[0033] **FIG. 12** shows one or more object structures following the card header of **FIG. 11**;

[0034] **FIG. 13** is a data flow diagram showing the flow of messages within the system of FIGS. **6**(a) and **6**(*b*);

[0035] **FIG. 14**(*a*) is a perspective view of still another smart card;

[0036] **FIG. 14**(*b*) shows user interface element objects associated with the smart card of **FIG. 14**(*a*);

[0037] **FIG. 15**(*a*) is a perspective view of still another smart card;

[0038] **FIG. 15**(*b*) shows user interface element objects associated with the smart card of **FIG. 15**(*a*);

[0039] **FIG. 16** is a flow diagram showing steps performed by a user in order to retrieve on-line music, over the smart card interface system of **FIG. 6**(*b*);

[0040] FIGS. **17**(*a*) to **17**(*e*) show a number of examples of display output available from the system of **FIG. 6**(*b*);

[0041] **FIG. 18** is a flow diagram showing a method of checking the smart cards of FIGS. **14**(a) and **15**(*a*), performed during the user process of **FIG. 16**;

[0042] **FIG. 19** is a flow diagram showing a method of scanning the touch panel of the reader of **FIG. 3**, performed during the user process of **FIG. 16**;

[0043] **FIG. 20** is a flow diagram showing an overview of events of the system of **FIG. 6**(*a*) during the user process of **FIG. 16**;

[0044] **FIG. 21** is a flow diagram showing processes performed by the event manager during the process of **FIG. 20**;

[0045] **FIG. 22** is a flow diagram showing an overview of a method performed by the launcher during the process of **FIG. 20**;

[0046] **FIG. 23** is a flow diagram showing a method of operating the smart card of **FIG. 1**(*b*) using the reader of **FIG. 3**;

[0047] **FIG. 24** is a flow diagram showing a process executed when a command is issued to a user interface card resident application executing on the smart card of **FIG. 1**(*a*);

[0048] **FIG. 25** is a flow diagram showing an initialisation process;

[0049] **FIG. 26** is a flow diagram showing a proprietary instruction process;

[0050] **FIG. 27** is a flow diagram showing an International Standards Organisation (ISO) instruction process;

[0051] **FIG. 28** is a flow diagram showing a process coordinate process;

[0052] **FIG. 29** is a flow diagram showing a user interface element handler process;

[0053] **FIG. 30** is a flow diagram showing a buffer input user interface element handler process;

[0054] **FIG. 31** is a flow diagram showing a buffer OK process;

[0055] **FIG. 32** is a flow diagram showing a pass-code checker process;

[0056] **FIG. 33** is a flow diagram showing a buffer cancel process;

[0057] **FIG. 34** is a flow diagram showing a buffer back-space process;

[0058] **FIG. 35** is a flow diagram showing a buffer append process;

[0059] **FIG. 36** is a flow diagram showing a standard user interface element process;

[0060] **FIG. 37** is a flow diagram showing a buffered user interface element process;

[0061] **FIG. 38** is a flow diagram showing a delegator user interface element process;

[0062] **FIG. 39** is a flow diagram showing an output object data process executed when the card of **FIG. 1**(*b*) outputs data to the reader of **FIG. 3**;

[0063] **FIG. 40** is a flow diagram showing a save state process;

[0064] **FIG. 41** is a flow diagram showing a restore state process;

[0065] **FIG. 42** is a flow diagram showing a select file process;

[0066] **FIG. 43** is a flow diagram showing a read binary process;

[0067] **FIG. 44** is a flow diagram showing a write binary process;

[0068] **FIG. 45** is a flow diagram showing an erase binary process;

[0069] **FIG. 46** is a flow diagram showing a get challenge process;

[0070] **FIG. 47** is a flow diagram showing an external authenticate process;

[0071] **FIG. 48** shows a smart card depicting an image of a human face;

[0072] **FIG. 49**(*a*) shows an example buffer descriptor; and

[0073] **FIG. 49**(*b*) is a perspective view of another smart card.

DETAILED DESCRIPTION INCLUDING BEST MODE

[0074] Where reference is made in any one or more of the accompanying drawings to sub-steps and/or features, which have the same reference numerals, those sub-steps and/or features have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

[0075] Excepting where explicitly distinguished, in the following description, the term "data string" means 'a sequence of bits (i.e. '1' or '0')' and can include American Standard Code for Information Interchange (ASCII) data, floating point data, and binary representations of integer values, for example.

[0076] The embodiments disclosed herein have been developed primarily for use with automatic tellers, remote control and network access systems, and will be described hereinafter with reference to these and other applications. The embodiments disclosed herein can be used to access services such as home shopping, home-banking, video-on-demand, interactive applications such as games and interactive trading cards, and information access such as city guides, television program guides and educational material. However, it will be appreciated that the invention is not limited to these fields of use.

[0077] For ease of explanation the following description has been divided into Sections 1.0 to 2.6, each section having associated subsections.

[0078] 1.0 SMART CARD INTERFACE SYSTEM OVERVIEW

[0079] 1.1 Smart cards

[0080] **FIG. 1**(*a*) shows a smart card **100**A including a planar substrate **112** with various user interface elements **114** (i.e. predetermined areas, or iconic representations) printed or otherwise formed on an upper face **116** thereof, for example using an adhesive label. For the smart card **100**A, the user interface elements **114** are in the form of a four way directional controller **120**, a "jump button"**122**, a "kick button"**124**, a "start button"**128** and an "end button"**130** printed on a front face **116** thereof. Other forms of user interface elements, such as promotional or instructional material, can be printed alongside the user interface elements **114**. For example, advertising material **126** can be printed on the front face **116** of the smart card **100**A or on a reverse face (not shown) of the smart card **110**A. In still other forms of the memory card **110**A, the memory chip **219** can be replaced by a storage means such as a magnetic strip (not shown) formed on one surface of the memory card **110**A.

[0081] As seen in **FIG. 2**(*a*), the front face **116** of the smart card **110**A may be formed by an adhesive label **260** upon which is printed the user interface in the form of the user interface elements **114**, in this case corresponding to the "End Button" and the Right arrow "button" of the directional controller **120**. The label **260** is affixed to the planar substrate **112**. A home user can print a suitable label for use with a particular smart card **100**A by using a printer. Alternatively, the user interface elements **114** can be printed directly onto the planar substrate **112** or separate adhesive labels can be used for each of the user interface elements **114**.

[0082] As also seen in **FIG. 2**(*a*), the smart card **100**A includes storage means in the form of an on-board memory chip **219** for storing data associated with the user interface elements **114**. The smart card **100**A having a memory chip **219** as described above is generally referred to as a "memory card". Thus, the smart card **100**A will hereinafter be referred to as the memory card **100**A. The memory card **100**A also includes electrical data contacts **218** connected to the memory chip **219** and via which reading of the memory chip **219** and writing to the memory chip **219** may be performed. Alternatively, in other forms of the memory card **100**A, the memory chip **219** can be replaced by storage means in the form of machine-readable indicia such as an optical code (e.g. a barcode) printed on a reverse face (not shown) of the memory card

[0083] Memory cards such as the memory card **100**A can be utilised in applications where strong security of the smart

card **100**A and data stored in the chip **219** of the smart card **100**A, is not required. The memory card **100**A can also be used in applications where it is desired to maintain the cost of manufacturing the smart card **100**A to a minimum. Such smart cards can be used for example, where the memory card **100**A is given away to promote a service and/or to provide access to the service. The memory card **100**A can also be used as a membership card, which provides access to a specific service.

[**0084**] **FIG. 1**(*b*) shows another smart card **100**B again including a planar substrate **152** with various user interface elements **154** printed on a front face **156** thereof. In the smart card **100**B the user interface elements **154** are in the form of a numerical keypad **160**, an "OK button"**162**, a "cancel button"**164**, a "clear button"**166** and a "backspace button"**168** printed on the front face **156** thereof. Again, other forms of user interface elements, such as promotional or instructional material, can be printed alongside the user interface elements **154** such as advertising material **158**.

[**0085**] As seen in **FIG. 2**(*b*), the front face **156** of the smart card **100**B is formed by an adhesive label **270** affixed to the planar substrate **152** in a similar manner to the memory card **100**A. Again, a user interface in the form of user interface elements **154**, in this case corresponding to the "number 3", "number 6" and "number 9" buttons of the numerical keypad **160** and the "backspace button"**168**, is printed on the adhesive label **270**.

[**0086**] As also seen in **FIG. 2**(*b*), the smart card **100**B includes a microprocessor **259** having an integrally formed central processing unit (CPU) **275** and storage means **276**. The storage means **276** generally includes volatile random access memory (RAM) (not shown) and non-volatile flash (ROM) memory (not shown), and can be used to store data associated with the user interface elements **154**, application software code associated with the smart card **100**B and/or information (e.g. a personal identification number) associated with the user and/or manufacturer of the smart card **100**B. The smart card **100**B will hereinafter be referred to as the CPU card **100**B. The CPU card **100**B also includes electrical data contacts **278** connected to the microprocessor **259** and which perform a similar role to the contacts **218** of **FIG. 2**(*a*). In particular, the electrical data contacts 278 can be used to send instructions to the microprocessor **259** and to receive data resulting from the execution of those instructions on the microprocessor **259**.

[**0087**] CPU cards such as the CPU card **100**B can be utilised in applications where enhanced user authentication and/or higher levels of security of the CPU card **100**B and data stored in the storage means **276**, is required.

[**0088**] It will be appreciated by a person skilled in the relevant art, that the user interfaces in the form of the user interface elements **114** and **154** can be interchanged for the smart cards **100**A and **100**B. Further, the user interfaces able to be printed by a user and/or manufacturer for the smart cards **100**A and **100**B can take many forms. Memory cards such as the memory card **100**A and CPU cards such as the CPU card **100**B, having a user interface formed on one surface of the card can be referred to as 'User Interface Cards. However, excepting where explicitly distinguished, in the following description, the memory card **100**A and the CPU card **100**B will be generically referred to as the smart card **100**.

[**0089**] 1.2 Smart card Reader

[**0090**] **FIG. 3** shows a smart card reader **300** configured for use with both the memory card **100**A and the CPU card **100**B. The configuration of the electrical data contacts **218** and **278** of the memory card **100**A and the CPU card **100**B, respectively, correspond to exposed contacts **307** of the smart card reader **300**, as shown in **FIG. 3**. The reader **300** is formed of a housing **301** incorporating a receptacle **304** into which the smart card **100** may be inserted, a viewing area **306** and an access opening **310** configured to accept a smart card **100**. An upper boundary of the viewing area **306** is defined by sensor means in the form of a substantially transparent pressure sensitive membrane **308** or simply "touch panel" spaced above the exposed contacts **307** so as to form the receptacle **304**. It will be appreciated by a person skilled in the relevant art that alternative technology can be used as the touch panel **308**. For example, the touch panel **308** can be resistive or temperature sensitive.

[**0091**] In use, a smart card **100** is inserted by a user into the smart card receptacle **304**, through the access opening **310**, as shown in **FIG. 4**. When the smart card **100** is fully inserted into the reader **300**, the touch panel **308** fully covers the upper face **116**, **156** of the smart card **100**. The viewing area **306** preferably has substantially the same dimensions as the upper face **116**, **156** of the smart card **100** such that the upper face **116**, **156** is, for all intents and purposes, fully visible within the viewing area **306** through the touch panel **308**. In this position, the data contacts **218**, **278** of the card **100** engage the exposed contacts **307** so that circuitry (not shown) within the reader **300** can communicate with the memory chip **219** or microprocessor **259** of the card **100**.

[**0092**] When the card **100** is fully inserted into the reader **300**, a user can press areas of the touch panel **308**, as shown in **FIG. 5**, overlying the user interface elements **114**, **154**. For the memory card **100**A, the reader **300** deduces which of the user interface elements **114** the user has selected by sensing the pressure on the touch panel **308** and referring to the data stored in the memory chip **219**. For example, if the user places pressure on the touch panel **308** adjacent the "kick button"**124**, the reader **300** is configured to assess the position at which the pressure was applied, refer to the stored data, and determine that the "kick button"**124** was selected.

[**0093**] In contrast, for the CPU card **100**B, the CPU **275** determines which of the user interface elements **154** the user has selected by processing coordinates received from the reader **300** upon the reader **300** sensing pressure on the touch panel **308**, and then the CPU **275** referring to the data stored in the storage means **276** of the microprocessor **259**. In this case, it is not necessary for the reader **300** to be able to read and to be made aware of the data stored in the storage means **276** of the microprocessor **259**. The operation of the CPU card **100**B in relation to the reader **300** will be explained in more detail in Sections 2.0 to 2.6 below.

[**0094**] Information resulting from a user selecting one of the user interface elements **114**, **154** can be used to control an external device, for example, an associated automatic teller machine (of conventional construction and not shown). It will be appreciated from above that the user interface elements **114**, **154** are not, in fact buttons. Rather, the user interface elements **114** are user selectable features which, by virtue of their corresponding association with the

data stored in the memory chip **219** or storage means **276**, and the function of the touch panel **308**, operate to emulate buttons traditionally associated with remote control devices.

[**0095**] In one advantageous implementation, the reader **300** includes a transmitter (of conventional type and not shown), such as an infra-red (IR) transmitter or radio frequency (RF) transmitter, for transmitting information in relation to user interface elements **114**, **154** selected by the user. In this case, upon selection of one of the user interface elements **114**, **154**, the reader **300** causes information related to the selection to be transmitted to a remote console (not shown in **FIG. 5**) where a corresponding infra-red or radio frequency remote module can detect and decode the information for use in controlling some function, such as a banking application executing on the automatic teller machine discussed above.

[**0096**] Any suitable transmission method can be used to communicate information from the reader **300** to a remote module, including direct hard wiring. Moreover, the remote module itself can incorporate a transmitter, and the reader **300** a receiver for communication in an opposite direction to that already described. The communication from the remote module to the reader **300** can include, for example, handshaking data, setup information, or any other form of information desired to be transferred from the remote module to the reader **300**.

[**0097**] **FIG. 10** is a schematic block diagram showing the internal configuration of the reader **300** in more detail. The reader **300** includes a microcontroller **1044** for controlling the reader **300**, for coordinating communications between the reader **300** and a remote module, and for storing mapping information and firmware, for example. The microcontroller **1044** includes random access memory (RAM) **1047** and flash (ROM) memory **1046**. The microcontroller **1044** also includes a central processor unit (CPU) **1045**. The microcontroller **1044** is connected to a clock source **1048** and a clock controller **1043** for coordinating the timing of events within the microcontroller **1044**. The CPU **1045** is supplied with electrical power from a battery **1053**, the operation of the former being controlled by a power controller **1050**. Alternatively, in one implementation, the CPU **1045** can be supplied with power via a universal serial bus cable (not shown) connected to a reader associated with the power coming from a personal computer or similar device. The microcontroller **1044** is also connected to a beeper **1051** for giving audible feedback about card entry status.

[**0098**] Infra-red (IR) communications, as discussed above, can be implemented using two circuits connected to the microcontroller **1044**, an infra-red transmitter (TX) **1049** for infra-red transmission and an infra-red remote module (RX) **1040** for infra-red reception. The touch panel **308** of the reader **300** communicates with the microcontroller **1044** via a touch panel interface **1041** and the electrical contacts **307**.

[**0099**] An in-system-programming interface **1052** can also be connected to the microcontroller **1044**, to enable programming of the microcontroller **1044** with firmware by way of the microprocessor flash memory **1046**.

[**0100**] 1.3 Hardware Architecture

[**0101**] **FIG. 6**(*a*) shows a hardware architecture of a card interface system **600**A. In the system **600**A, the reader **300** is hard wired to a personal computer system **700** via a communications cable **603**. Alternatively, instead of being hardwired, a radio frequency or infrared transceiver **654** formed in the personal computer system **700** can be used to communicate with the reader **300**. The personal computer system **700** includes a display device **701**, a computer module **702**, a keyboard **704** and mouse **703**, and will be explained in more detail below with reference to **FIG. 7**.

[**0102**] The system **600**A includes the smart card **100** which is programmable and can be created or customised by a third party, who may be a party other than the manufacturer of the smart card **100** and/or the card reader **300**. The third party can be the ultimate user of the smart card **100** itself, or may be an intermediary between the manufacturer and user. In the system **600**A of **FIG. 6**(*a*), the smart card **100** can be programmed and customised for one touch operation to communicate with the computer **700** and obtain a service over a computer network **720**, such as the Internet, coupled to the computer **700**. The computer **700** operates to interpret signals sent via the communications cable **603** from the reader **300**, according to a specific protocol, which will be described below. The computer **700** performs the selected function according to touched user interface elements **114**, **154** and can be configured to communicate data over the network **720**. In this manner, the computer **700** can permit access to applications and/or data stored on remote server computers **650**, **652** and appropriate reproduction on the display device **701**, by way of user manipulation of the reader **300** and card **100**.

[**0103**] **FIG. 6**(b) shows a hardware architecture of a card interface system **600**B. In the system **600**B, the reader **300** can be programmed for obtaining a service locally at a set top box **601**, that couples to an output interface, which in this example takes the form of an audio-visual output device **616**, such as a digital television set. The set-top box **601** operates to interpret signals **612** received from the reader **300**, which may be electrical, radio frequency, or infra-red (IR), and according to a specific protocol which will be described below. The set top box **601** can be configured to perform the selected function according to touched user interface elements **114**, **154** and permit appropriate reproduction on the output device **616**. Alternatively, the set top box **601** can be configured to convert the signals **612** to a form suitable for communication and cause appropriate transmission to the computer **700** via the network **720**. The computer **700** can then perform the selected function according to the user interface elements **114**, **154**, and provide data to the set-top box **601** to permit appropriate reproduction on the output device **616**. The set top box **601** will be explained in more detail below with reference to **FIG. 8**.

[**0104**] In one application of the system **600**B, the smart card **100** can be programmed for obtaining a service either remotely or locally. For instance, the smart card **100** can be programmed to retrieve an application and/or data stored on remote server computers **650**, **652**, via the network **720**, and to load the application or data on to the set top box **601**. **619191**US.doc The latter smart card can be alternatively programmed to obtain a service from the loaded application on the set top box **601**.

[**0105**] Excepting where explicitly distinguished, the systems **600**A and **600**B of **FIG. 6**(*a*) and **6**(*b*) will be hereinafter generically referred to as the system **600**.

[0106] **FIG. 7** shows the general-purpose computer system **700** of the system **600**, which can be used to run the card interface system and to run software applications for programming the smart card **100**. The computer system **700** includes the computer module **702**, input devices such as the keyboard **704** and mouse **703**, output devices including a printer (not shown) and the display device **701**. A Modulator-Demodulator (Modem) transceiver device **716** is used by the computer module **702** for communicating to and from the communications network **720**, for example connectable via a telephone line **721** or other functional medium. The modem **716** can be used to obtain access to the Internet, and other network systems, such as a Local Area Network (LAN) or a Wide Area Network (WAN).

[0107] The computer module **702** typically includes at least one central processing unit (CPU) **705**, a memory unit **706**, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), input/output (I/O) interfaces including a video interface **707**, and an I/O interface **713** for the keyboard **704** and mouse **703**, a write device **715**, and an interface **208** for the modem **216**. The I/O interface **713** also includes the IR transceiver **654** connected to the I/O interface **713** for communicating directly with the reader **300**. A storage device **709** is provided and typically includes a hard disk drive **710** and a floppy disk drive **711**. A magnetic tape drive (not illustrated) is also able to be used. A CD-ROM drive **712** is typically provided as a non-volatile source of data. The components **705** to **713** of the computer module **702**, typically communicate via an interconnected bus **704** and in a manner, which results in a conventional mode of operation of the computer system **702** known to those in the relevant art. Examples of computers on which the arrangement described herein can be practised include IBM- computers and compatibles, Sun Sparcstations or alike computer system evolved therefrom.

[0108] Typically, the software programs such as applications of the system **600** are resident on the hard disk drive **710** and read and controlled in their execution by the CPU **705**. Intermediate storage of the software application programs and any data fetched from the network **720** may be accomplished using the semiconductor memory **706**, possibly in concert with the hard disk drive **710**. In some instances, the application programs can be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive **712** or **711**, or alternatively may be read by the user from the network **720** via the modem device **716**. Still further, the software can also be loaded into the computer system **702** from other computer readable medium including magnetic tape, ROM or integrated circuits, a magneto-optical disk, a radio or infra-red transmission channel between the computer module **702** and another device, a computer readable card such as a smart card, a computer PCMCIA card, and the Internet and Intranets including email transmissions and information recorded on Websites and the like. The foregoing is merely exemplary of relevant computer readable media. Other computer readable media are able to be practised without departing from the scope of the invention defined by the appended claims.

[0109] **FIG. 8** shows the set top box **601** of the system **600**, which can be used to interpret the signals **612** received from the reader **300**. The set top box **601** in some implementations essentially is a scaled version of the computer module **702**. The set top box **601** typically includes at least one CPU unit **805**, a memory unit **806**, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), and input/output (I/O) interfaces including at least an I/O interface **813** for the digital television **616**, an I/O interface **815** having an IR transceiver **808** for receiving and transmitting the signals **612**, and an interface **817** for coupling to the network **720**. The components **805**, **806**, **813**, **815** and **817** of the set top box **601**, typically communicate via an interconnected bus **804** and in a manner which results in a conventional mode of operation. Intermediate storage of any data received from the reader **300** or network **720** may be accomplished using the semiconductor memory **806**. Alternatively, the set top box can include a storage device (not shown) similar to the storage device **709**.

[0110] 1.4 Programming the Smart card

[0111] As described above, the smart card **100** is programmable and can be created or customised by a third party. For example. with the system **600**, the smart card **100** can be programmed and customised for one touch operation to communicate with the set-top box **601** and/or computer **700** and obtain a service over the network **720**. The smart card **100** can be programmed by means of the write device **715** coupled to the I/O interface **713** of the computer module **702**. The write device **715** has the capability of writing data to the memory chip **219** on the memory card **100**A or the storage means **276** of the microprocessor **259** for the CPU card **100**B. Preferably, data is not able to be written to the storage means **276** unless a value, calculated in accordance with a predetermined electronic key, is first presented to the microprocessor **259**. Depending upon the specific implementation the write device **715** may also be configured to print graphics on to the front surface **116**, **156** of the smart card **100** using image production software application programs. The write device **715** may also have a function for reading data from the smart card **100**.

[0112] The write device can be configured such that the user can insert the smart card **100** into the write device **715** and then enter the required data. A software application can then write the data entered by the user to the memory of the smart card **100** via the write device **715**. If the stored data is encoded for optical decoding such as in the case of a barcode memory card, the write device **715** can print the encoded data onto the memory card **100**A.

[0113] For the CPU card **100**B, the microprocessor **259** can be constructed so that once programmed in the manner described, the contents cannot thereafter be casually read.

[0114] 1.5 Software Architecture Layout

[0115] **FIG. 9** shows a software architecture **900** for the hardware architecture depicted by the system **600**. The architecture **900** can be divided into several distinct process components and one class of process. The distinct processes include an I/O interface **920**, which may be colloquially called an "I/O daemon"**920**, an event manager **901**, a display manager **906**, an (application) launcher **903** and a directory service **911**. The class of process is formed by one or more applications **904**. In the architecture **900** described herein, there exists one I/O daemon **920**, one event manager **901**, one display manager **906** and one launcher **903** for every smart card remote connection, usually formed by the set-top box **601**. Where processes responsible for multiple remote

connections are running on a single computer (e.g. the computers **700**, **650** or **652**) which provides support for a number of set-top boxes **601**, a master launcher **908** (shown in **FIG. 9** in phantom lines) can be used. The master launcher **908** communicates directly with the event manager **901** and starts a launcher **903** for every set-top box **601** which connects to master launcher **908**. The architecture **900** also includes at least one directory service **911**. The directory service **911**, is queried by the launcher **903** to translate data into a Resource Locater (eg. URL) that indicates a name or location of a service or the location or name of an application **904** to be used for the service.

[**0116**] In this form, the architecture **900** can be physically separated into six distinct parts **601**, **701**, **907**, **909**, **912** and **913** as shown by the dashed lines in **FIG. 9**, each of which can be run on physically separate computing devices. Alternatively, the display **701** can be a device with little or no computing capacity. Communication between each of the parts of the system **900** can be executed using Transport Control Protocol/ Internet Protocol (TCP/IP) streams. Alternatively, each of the parts **601**, **701**, **907**, **909**, **912** and **913** can be run on the same machine.

[**0117**] In the system **600**A of **FIG. 6**(*a*), all of the process components **901**, **903**, **904**, **906** and **920** can be run on the computer **700**. The event manager **901**, the launcher **903** and display manager **906** are preferably integrated into one executable program which is stored in the hard disk **709** of the computer **700** and can be read and controlled in its execution by the CPU **705**. The directory service **911** may run on the same computer **700** or on a different computer (e.g. server **650**) connected to the computer **700** via the network **720**.

[**0118**] In the system **600**B of **FIG. 6**(*b*), all of components **901**, **903**, **904**, **906** and **920** can run from the set-top-box **601**. In this instance, the components **901**, **903**, **904**, **906** and **920** can be stored in the memory **806** of the set top box **601** and can be read and controlled in their execution by the CPU **805**. The directory service **911** can run on one of the computers **650**, **652** or **700** and can be stored in the hard disk drive **710** of the computer **700**, for example, and be read and controlled in its execution by the CPU **705**. Alternatively, the directory service **911** can be run on the set top box **601** or its function executed by the launcher **903**. In a further alternative arrangement, if the set-top-box **601** is not powerful enough to run the system **600** locally, only the I/O daemon **920** need run on the set-top-box **601** and the remainder of the architecture **900** (i.e. process components **901**, **903**, **904**, **906** and **911**) can be run remotely on the other servers (**650**, **652**) which can be accessed via the network **720**. In this instance, the I/O daemon **920** can be stored in the memory **806** of the set top box **601** and can be read and controlled in its execution by the CPU **805**. Again, the functional parts of such a system can be divided as shown in **FIG. 9**. In this instance, the display **701** corresponds to an audio-visual output device (e.g. a television set).

[**0119**] The I/O daemon **920** is a process component that converts datagrams received from the reader **300** into a TCP/IP stream that can be sent to the event manager **901** and vice versa (e.g. when using a two-way protocol). The event manager **901** is configured to gather all events that are generated by the reader **300** and relayed by the I/O daemon **920**. These events are then redistributed to the various process components **903**, **904**, **906** and **908** and running applications. The event manager **901** is also configured to check that an event has a valid header and correct data length. An "event" in this regard represents a single data transaction from the I/O daemon **920** or the launcher **903** or applications **904**. The master launcher **908** can be used to start the launcher **903** corresponding to each of the event managers **901** if more than one event manager is running on the system **600**. The launcher **903** is an application that starts other applications for a specific event manager **901**. The launcher **903** starts and ends applications and can also start and end sessions. The launcher **903** also informs the event manager **901** when applications are starting and ending, and informs the applications **904** when they are gaining or losing focus, or when they need to exit. The display manager **906** selects which smart card application **904** is currently able to display output on the display device **701** (i.e. the "front" application).

[**0120**] The directory service **911** is configured to translate service identifiers that are stored on smart cards **100**, into resource locators (e.g. a URL) that indicate the location of the services or the location of an application associated with a service. The directory service **911** is also configured to translate optional service data. The directory service **911** allows the launcher **903** associated with a particular card **100** to decide what to do with a resource locator, for example, download and run the associated application **904** or load the resource locator into a browser application.

[**0121**] The applications **904** associated with a particular smart card **100** can be started by the launcher **903** associated with that smart card **100** in response to a selection of one of the user interface elements **114**, **154** of a corresponding smart card **100**. Each application **904** can be a member of one or more application service groups. An application service group is comprised of a number of smart card applications **904** that act co-operatively, as opposed to merely simultaneously, to provide a particular set of functions. An application **904** can be specified to not be part of any service group in which case the application will never be run with other applications. An application can become part of a service group once the application is running and can remove itself from a service group when the application is the currently front application.

[**0122**] In a still further alternative arrangement, the process components **901** to **906** and **908** described above can be implemented in dedicated hardware (e.g. the set top box **601**) as one or more integrated circuits performing the described functions or sub-functions. Such dedicated hardware may include graphic CPUs, digital signal CPUs, or one or more micro-CPUs and associated memories.

[**0123**] Typically, applications **904** are resident on the hard disk drive **710** and read and controlled in their execution by the CPU **705**. Intermediate storage of programs and any data fetched from the network **720** can be accomplished using the semiconductor memory **706**, possibly in concert with the hard disk drive **710**. In some instances, the applications **904** will be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive **711** or **712**, or alternatively may be read from the network **720** via the modem device **716**. Other mechanisms for loading the applications **904** into the computer system **700** from other computer readable medium include magnetic tape, a ROM

or integrated circuit, a magneto-optical disk, a radio or infra-red transmission channel between the computer module **702** and another device, a computer readable card such as a smart card (not shown), a computer 'Personal Computer Memory Card International Association (PCMCIA) card' (not shown), and the Internet and/or Intranets including email transmissions and information recorded on Websites and the like. The foregoing is merely exemplary of relevant computer readable media. Other computer readable media are also possible including combinations of those described above .

[**0124**] 1.6 Smart card data format

[**0125**] The smart card **100** generally stores a data structure in memory **219**, **276** that describes various card properties and any user interface elements **114**, **154** printed on the smart card **100**. The smart card **100** can also include global properties that specify attributes such as information about the smart card **100**, vendor and a service. Further, user-interface objects, as will be explained in detail below, can specify data to be associated with the user interface elements **114**, **154** printed on the surface of a corresponding smart card **100**.

[**0126**] For the memory card **100**A, data conforming to the format to be described can be copied directly into the memory chip **219** of the smart card **100**. For the CPU card **100**B, data conforming to the format to be described can be stored in the storage means **276** as a file being one file of a file system implemented on the CPU card **100**B. Such a file system will be described in detail below. In either case, to ensure that the cost of the smart card **100** can be kept to a minimum, the amount of data stored on the smart card **100** is kept to a minimum. For example, where the smart card **100** is being used as a music sampler and associated on-line service, the memory **219**, **276** of the smart card **100** does not contain the music itself. The smart card **100** only contains data associated with the user interface in the form of the user interface elements **114**, **154** and certain identifiers, which will be described in detail below. If the smart card **100** has limited storage capacity (e.g. in the case where the smart card **100** utilises a barcode), the smart code **100** may only include a card identifier as will be explained in detail below.

[**0127**] The user-interface objects referred to above can represent mapping data, which relate the user interface elements **114**, **154** imprinted directly on a surface of the smart card **100**, to commands or addresses (eg: Uniform Resource Locators (URLs)). The mapping data includes (X,Y) coordinates that typically define the size and location of user interface elements **114**, **154** on the smart card **100**. The user-interface objects are preferably stored in the memory **219**, **276** of the smart card **100**. Alternatively, the user-interface objects can be stored not on the smart card **100** itself, but in the system **600**. For example, the smart card **100** can store, via the memory **219**, **276** a barcode or a magnetic strip, a unique identifier, which is unique to smart cards **100** having substantially similar user interface elements **114**, **154** and layout. The unique identifier together with the coordinates determined from the touch panel **308**, as a result of a user press, can be transmitted by the reader **100** to the computer **700** or to the set top box **601**, of the system **600**.

[**0128**] The system **600** can have the user-interface objects stored on the computer **700**, set top box **601** or server **650**, which may thus be arranged to perform the mapping from the determined coordinates to a corresponding command, address or data relevant to a service associated with the smart card **100** and a user selection of one of the user interface elements **114**, **154**, in order to provide a desired function represented by the selected user interface elements. In this instance, the data related to the user selected user interface elements **114**, **154** as described above takes the form of coordinates determined by the microcontroller **1044** of the reader **300** as a result of a user applying pressure to a portion of the touch panel **308** which overlays the desired user interface elements **114**, **154**.

[**0129**] Data stored in the smart card **100** includes a card header followed by zero or more objects as described in the following sections.

[**0130**] 1.6.1 Card Header

[**0131**] **FIG. 11** shows the data structure of a card header **1100** as stored in the smart card **100**. The header **1100** includes a number of rows **1101**, each of which represent four bytes of data. The data is preferably in 'big endian' format. The complete header is 19 bytes long and includes the following fields (described in more detail in Table 1 below):

[**0132**] (i) magic number field **1102**, which includes a constant specifying a smart card as being a valid memory card **100**A or CPU card **100**B. For example, the magic number field **1102** can be used to check or verify that a propriety card belonging to a particular manufacture is being used;

[**0133**] (ii) versions field **1103**, which includes each version increment that specifies a change in the smart card layout that cannot be read by a reader **300** which is compatible with lower versions of the layout;

[**0134**] (iii) reserved field **1104**, this field is reserved for future use;

[**0135**] (iv) flags field **1105**, which includes flags for a smart card (see Table 2 below);

[**0136**] (v) card identifier field **1110**, which includes two fields—a service **1106** and a service specific field **1107**. The service field **1106** identifies the service of a corresponding smart card **100** and the service specific field **1107** optionally contains a service-specific value;

[**0137**] (vi) a number of objects field **1108**, which includes a number value representing how many objects follow the header **1100**. This field can be set to zero; and

[**0138**] (vii) a checksum field **1109**, which includes a card checksum of all data on a corresponding smart card **100** excluding the checksum itself.

[0139] Table 1 below provides a description of the content of the various (number) fields described with reference to **FIG. 11**.

TABLE 1

Field Number	Description (Card Header)
Magic Number	Two byte magic number. A constant that specifies this as being a valid card. Currently defined as the ASCII value for 'i' followed by the ASCII value for 'C'.
Version	One byte version number. Each version increment specifies a change in the card layout that can not be read by a reader that is compatible with lower versions of the layout. This document describes version 1(0x01) of the card format.
Reserved	This data is reserved for future use. Its value must be set to zero.
Flags	Four bytes of flags for this card. (See Table 2). All non-assigned bits must be zero.
Card Identifier	Eight byte card identifier. Card identifiers include two fields—service identifier and service-specific identifier. The service identifier is five bytes and identifies the service associated with the card. The service-specific identifier is three bytes of service specific value.
Number of Objects	One byte. The number of objects following this header. Can be zero.
Checksum	Card checksum, 2 bytes. The card checksum is sixteen bit, unsigned integer sum of all data bytes on the card excluding the checksum.

[0140] The card identifier field **1110** comprises an eight-byte card identifier. The card identifier includes two portions (i.e. unit pieces of data), namely, a service identifier and a service-specific identifier. Preferably, the card identifier is arranged so that the service identifier occupies five bytes and the service-specific identifier occupies three bytes of the total card identifier value.

[0141] The service identifier contained in the field **1106** may be used to distinguish one service from another or distinguishes one vendor from another. That is, for example, a service can be associated with an application that provides the service to the user of a smart card **100** as distinct from a vendor who can provide multiple services to the user by providing multiple applications. The service identifier can be an identifier to identify the application to be used or application location (e.g. URL).

[0142] The card identifier can be used to configure generic smart cards for the system **600**. Generic smart cards are smart cards **100** having a special service identifier that can be used to provide input to a current application already running. The special value for the service identifier, referred to as "the generic service identifier", is 0×0000000001, where '0×' represents hexadecimal notation (i.e. every two characters of the generic service identifier represent the value of a single byte). A generic smart card can be used to send data to a front application already running on the system **600**. For example, a smart card **100** having a keypad user interface that can be used to send text input to an application which has focus or a smart card **100** with personal details that can also be used to submit the personal information stored on the smart card **100** to any application.

[0143] A smart card **100** identification authority can assign service identifiers to a vendor when the vendor registers a particular service.

[0144] The service-specific identifier contained in the field **1107**, as described above, can be optionally used by the vendor of a particular service to provide predetermined functions associated with that particular service. The use of the service-specific identifier is substantially dependent upon an application **904** being executed on the system **600**. For example, the service identifier together with the service-specific identifier can be used as a unique identifier for a card **100**. This unique identifier can be used by an application **904** to gain or deny access to a specific feature associated with a particular service, to reproduce a specific-service identifier in a log file in order to confirm or verify that a particular smart card **100** having that value was used to access a service, and to provide a unique identifier that can be matched up with a corresponding value in a database in order to retrieve information about the user of the service (eg: name, address, credit card number etc).

[0145] Another example of a use for the service-specific identifier can include providing information about a mechanism or mode of distribution of the smart cards **100** (e.g. by mail, bus terminal kiosks, handed out on a train etc). Further, the service-specific identifier, can identify what data is to be loaded into the system **600** when a service is accessed.

[0146] The foregoing is not intended to be an exhaustive list of possible uses or applications of the service-specific identifier but a small sample of possible applications and there are many other applications of the service-specific identifier of field **1107**.

[0147] **1.6.2** Card Flags

[0148] The flags field **1105** of the header **1100** of **FIG. 11** can include three flags.

[0149] For the memory card **100**A and the CPU card **100**B, the flags of the flags field **1105** are as follows:

[0150] (i) Background beep;

[0151] (ii) Move; and

[0152] (iii) Don't Report Background Coordinates.

[0153] Table 2 below provides a description of each of the above flags (i) to (iii). The above flags (i) to (iii) affect the functions that the smart card 100 can perform in a reader **300**, as is defined by the description of each flag.

TABLE 2

Name	Description	Value (hex)
Background Beep	Causes the reader to provide audio feedback whenever the background is touched.	0x0000 0001
Move	Causes the reader to send all move events from when the touch panel was pressed until the touch panel is released.	0x0000 0002
Don't Report Background Co-ordinates	Causes the reader to suppress reporting of the co-ordinates of all presses and releases of the touch panel, when they correspond to the background, reporting them instead as (0xFF, 0xFF).	0x0000 0004

[0154] 1.6.3 Objects

[0155] As shown in **FIG. 12**, immediately following the card header **1100** of **FIG. 11** can be zero or more object structures **1213** defining the objects of a particular smart card **100** and forming part of the data stored on the smart card **100**. Each object structure **1213** comprises four fields as follows:

[0156] (i) a type field **1201**;

[0157] (ii) an object flags field **1203**;

[0158] (iii) a length field **1205**; and

[0159] (iv) a data field **1207**.

[0160] The structure of the data field **1207** depends on the object type as will be described below.

[0161] Table 3 below shows a description of each of the fields **1201**, **1203**, **1205** and **1207** of the object structure **1213**.

TABLE 3

Name	Description (Object Structure)	Length
Type	The type of object (see Table 5).	1 byte
Object Flags	The general object flags that are associated with this object (see Table 4). Note: Additional flags specific to an object type are specified within the data field of the object.	1 byte
Length	The length of the data following this object. This value can be zero.	2 bytes
Data	The data associated with this object. The structure of this data is dependent on the type of object.	Variable

[0162] The flags field **1203** of the object structure **1213**, preferably includes an inactive flag. Table 4 below shows a description of the inactive flag.

TABLE 4

Name	Description (Pre-Object Flag Values)	Value (hex)
Inactive	Indicates to the reader that the object is valid but is to be ignored regardless of it's type.	0x01

[0163] For the smart card **100**, there are preferably eight object types provided, as follows:

[0164] (i) User Interface Element: Text;

[0165] (ii) User Interface Element: Delegator;

[0166] (iii) User Interface Element: Buffer

[0167] (iv) Card Data;

[0168] (v) Fixed Length Data;

[0169] (vi) Reader Insert;

[0170] (vii) No operation; and

[0171] (viii) No operation (single byte).

[0172] Table 5 shows a description of each of the above object types (i) to (viii).

TABLE 5

Name	Description)	Value (hex)
No operation (single byte)	A single byte object that doesn't have a standard object header. Used to fill spaces on the card that are too small for a normal object header.	0x00
No Operation	An object that is used to fill blocks of empty space on the card.	0x01
User Interface	A user interface element with raw	(inline) 0x10

TABLE 5-continued

Name	Description)	Value (hex)
Element: Text	data.	(file) 0x11
User Interface	A user interface element which	(inline) 0x12
Element: Buffer	initiates buffered input mode. (CPU card only).	(file) 0x13
User Interface	A user interface element containing	(inline 0x14
Element: Delegator	a command to be invoked on another card resident application (CPU card only).	(file) 0x15
Card Data	Contains data that relates specifically to this card. Card data would normally be read by the reader and sent as part of the INSERT message on card insertion.	0x20
Fixed length Data	An object that can be used to store fixed length blocks of data on the card.	0x30
Reader Insert	An object that can be used to give instructions to the reader when the card is inserted.	0x40

[0173] 1.6.3.1 User Interface Element Objects

[0174] Each of the user interface element objects of Table 5 define a rectangular area on a smart card **100** and some quantity of associated data that is is used to generate an output when the user touches an area of the touch panel **308** over the corresponding rectangular area of the smart card **100**. The origin for the coordinate mapping system is the top left of the smart card **100** in accordance with an International Standards Organisation standard smart card held in a portrait view with the chip contacts **218**, **278** facing away from the viewer and towards the bottom of the smart card **100**. For any reader **300** that does not use this card orientation, the values of corner points on the smart card **100** must be adjusted by the reader **300** for the memory card **100**A or by the CPU **275** for the CPU card **100**B, so as to report a correct "button" press.

[0175] The user interface element objects structure preferably has six fields as follows:

[0176] (i) a flags field;

[0177] (ii) an X1 field;

[0178] (iii) an Y1 field;

[0179] (iv) an X2 field;

[0180] (v) a Y2 field; and

[0181] (vi) a data field which typically includes data associated with the user interface element for example, a URL, a command, a character or name.

[0182] Table 6 shows a description of each of the above fields for the described user interface element object structure. A press on the touch panel **308** is defined to be inside an area defined by a particular user interface element corresponding to a user interface element object structure if:

[0183] (i) the X value of the press location is greater than or equal to the X1 value of the associated user interface element object and is strictly less than the X2 value for that particular user interface element object; and

[0184] (ii) the Y value for the press location is greater than or equal to the Y1 value of the particular user interface element object and strictly less than the Y2 value.

TABLE 6

Field	Description (User Interface Object Structure)	Size
Flags	Flags specific to this user interface element on the card.	1 byte
X1	X value of the top-left hand corner co-ordinate of this object's rectangle.	1 byte
Y1	Y value of the top-left hand corner co-ordinate of this object's rectangle.	1 byte
X2	X value of the bottom-right hand corner co-ordinates of this object's rectangle.	1 byte
Y2	Y value of the bottom-right hand corner co-ordinate of this object's rectangle.	1 byte
Data	Zero or more bytes of data associated with this object. In memory cards, the actual data is always stored within this field. For CPU cards, this field may contain a file identifier which points to a file where the data is actually stored. The size of this field is determined by the object data size minus the combined size of the above fields.	Variable

[0185] Overlapping user interface elements are allowed. In this case, if a press is within the bounds of more than one user interface element then the object resulting from the press is determined by a Z order. The order of the user interface elements **114**, **154** on the smart card **100** defines the Z ordering for all of the user interface elements on that particular smart card **100**. The top user interface element is the first user interface element for a particular smart card **100**. The bottom user interface element is the last user interface element for that particular smart card **100**. This allows for non-rectangular areas to be defined. For example, to define an "L" shaped user interface element, a first user interface element object would be defined with zero bytes in the data field, and a second user interface element object would be defined to the left and below the first user interface element object but overlapping the first user interface element object. The second user interface element would contain the data that is to be associated with the "L" shaped user interface element.

[0186] The location of a press is reported in "fingels", which represent finger elements (analogous to "pixels" which represent picture elements). The height of a fingel is defined to be 1/256th of the length of an International Standards Organisation memory smart card and the width is defined to be 1/128th of the width of an International Standards Organisation memory smart card.

[0187] The behaviour associated with each user interface element **114**, **154** may be modified using one or more flags. For both the memory card **100**A and the CPU card **100**B, each user interface element **154** preferably has seven associated flags as follows:

[0188] (i) Beep;

[0189] (ii) Move;

[0190] (iii) Don't report coordinates;

[0191] (iv) Auto repeat;

[0192] (v) Do Not Send Data on Press;

[0193] (vi) Do Not Send Data on Release; and

[0194] (vii) Encrypt Out-going data.

[0195] Table 7 shows a description for each of the user interface element flags (i) to (vii).

TABLE 7

Name	Description	Value
Beep	This flag causes the reader to beep when the user interface element is pressed.	0x01
Don't Report Co-ordinates	This flag instructs the reader to suppress reporting of the co-ordinates of the associated press or release, reporting them instead as (0xFF, 0xFF).	0x04
Auto-repeat	This element automatically repeats when the press is held on the element.	0x08
Don't Send Data on Press	This causes the associated user interface element not to send the data associated with this user interface element in the press event. The default is to send the data associated with the user interface element in the press event.	0x10
Don't Send Data on Release	This causes this user interface element not to send the data associated with this user interface element in the release event. The default is to send the data associated with the user interface element in the release event.	0x20
Encrypt Outgoing Data	This causes the data associated with the user interface element to be encrypted using a previously agreed upon session key. If no session key is present, no data is sent. (CPU cards only).	0x40

[0196] Data associated with a user interface element **114**, **154** on the smart card **100** can be referenced as 'Inline Data' which is stored directly in the data field of a user interface element. However, for the CPU card **100**B, data associated with a user interface element can also be referenced as 'File Data'. File data is stored in a separate elementary file in the storage means **276** of the microprocessor **259** and the structure of data field associated with such an elementary file is shown in Table 8.

TABLE 8

Name	Length	Description
File_id	2 bytes	The 16-bit ISO identifier corresponding to the required file
Header	variable	A variable length header, which is prepended to the file contents in order to identify the user interface element. This may be empty.

[0197] 1.6.3.2 Card Data

[0198] The card data object is used to store data which is specific to a particular card **100**. The data layout for this object has no fixed form. The contents of the card data object are sent from the reader **300** as part of an INSERT message when the smart card **100** is inserted into the reader **300**.

[0199] **1.6.3.4** Fixed Length Data

[0200] The fixed length data object is used to define a fixed length block on the smart card **100** that can be written to by the computer **700**, for example.

[0201] 1.6.3.5 Reader Insert

[0202] The reader insert object is used to store instructions for the reader **300** when a particular smart card **100** is inserted. This can be used, for example, to instruct the reader

300 to use a specific configuration of infra-red commands to allow communication with a specific set top box (e.g. **601**) or TV.

[**0203**] 1.6.3.6 No Operation

[**0204**] The No Operation object is used to fill in unused sections between other objects on a particular smart card **100**. Any data stored in the no operation object is ignored by the reader **300**. Any unused space at the end of the smart card **100** does not need to be filled in with a no operation object.

[**0205**] 1.6.3.7 No Operation (One Byte)

[**0206**] The No Operation (One Byte) object is used to fill gaps between objects that are too small for a full object structure. These objects are only one byte long in total.

[**0207**] 1.7 Reader Protocol

[**0208**] The reader **300** uses a datagram protocol that supports both uni-directional and bi-directional communication between the reader **300** and the set top box **601** or computer **700**, for example. The format used for messages from the reader **300** as a result of user interactions with the smart card **100** are of a different format than those that are sent to the reader **300**.

[**0209**] **1.7.1** Message Types

[**0210**] There are at least seven message event types that can be sent by the reader **300**. These message events are as follows:

[**0211**] (i) INSERT: When a smart card **100** is inserted into the reader **300**, and the smart card **100** is validated, an INSERT event is generated by the reader **300** and an associated message is transmitted. This message announces the smart card **100** to a remote module (e.g. the set top box **601**). The INSERT message preferably can include the particular card identifier and allow applications to be started or fetched immediately upon the smart card **100** insertion rather than waiting until the first interaction takes place. The INSERT message preferably includes the contents of the card data object from the smart card **100** inserted into the reader **300** if an object of this type is present on the smart card **100**.

[**0212**] (ii) REMOVE: When a smart card **100** is removed from the reader **300**, a corresponding REMOVE event is generated and a REMOVE message is sent to the particular remote module associated with the reader **300**. Like the INSERT message, the associated card identifier can be transmitted along with the message. As the card identifier cannot be read from the now removed smart card **100**, the card identifier is stored in the memory **1047** of the reader **300**. This is a useful optimisation as the card identifier is required for all other messages and reading the card identifier from the smart card **100** each time the card identifier is required can be too slow. INSERT and REMOVE messages are not relied upon by the system **600** to control processing. The system **600** is configured to infer missing messages if a message is received and is not immediately expected. For example, if an application detects two INSERT messages in a row, then an application can assume that it has missed the REMOVE message associated with the smart card **100** of the first INSERT message, as typically two smart cards **100** are not inserted into the reader **300** at one time. The application can then take whatever action is required prior to processing the second INSERT message.

[**0213**] Another example of where a missing message can occur is where a hand-held, infrared connected reader **300** as shown in **FIG. 6**(*b*), as compared with a wired reader **300** as shown in **FIG. 6**(a), is being used. Often a user does not point the reader **300** directly at a remote module when inserting or removing cards. This problem can be corrected by the system **600** inferring the INSERT or REMOVE operations based on differing card identifiers in consecutive PRESS and RELEASE pairs.

[**0214**] (iii) BAD CARD: If an invalid smart card **100** is inserted, then the reader 300 is preferably configured to generate a BAD CARD event and to send a BAD CARD message. This message allows an associated remote module to take some action to alert the user to the invalid smart card **100**.

[**0215**] (iv) PRESS: When a touch is detected by the reader **300**, a PRESS event is generated and a PRESS message is sent to an associated remote module. The PRESS message can contain details of an associated memory card, the position of the press and the data associated with the user-interface element at that particular position. If there is no user interface element defined for that position (including if there is no user interface element defined on the smart card **100** at all) a PRESS message is sent containing details of the associated smart card **100** and the position of the press. If there is no card present in the reader **300** when a PRESS event is generated then a PRESS message is sent containing the special "NO_CARD" identifier (i.e. eight bytes of zero - OxOO) and the- position of the press.

[**0216**] (v) RELEASE: A RELEASE event complements the PRESS event and a RELEASE message can be sent in order to inform an application program of the system **600** that a PRESS has been lifted. Every PRESS event preferably has a corresponding RELEASE event. Readers can allow multiple presses to be registered or provide other events that may occur between PRESS and RELEASE messages.

[**0217**] (vi) MOVE: If, after processing a PRESS event, the touch position changes by a certain amount then the finger (or whatever is being used to touch the smart card 100) is assumed to be moving. MOVE EVENTS are generated and MOVE messages are sent until the touch is lifted. MOVE events auto-repeat by re-sending the last MOVE messages when the touch position remains stationary. The repeated sending finishes when the touch is lifted and a corresponding RELEASE message is sent. Unlike PRESS and RELEASE events there is no user-interface object involved with MOVE events.

[**0218**] (vii) LOW BATT: A LOW BATT event is generated and a LOW BATT message is sent when the battery **1053** in the reader **300** is getting low. This message is sent after user interactions to increase the

chance that the message will be received by the rest of the system **600**. The sending of the LOW BATT message does not prevent the reader 300 from entering a low power state.

[**0219**] As described above, the card identifier is included in every INSERT, REMOVE, PRESS, RELEASE and MOVE message sent from the reader **300** to the computer **100** or set-top box **601**. As an alternative, the card identifier can be sent in connection with an INSERT message only. In this instance, upon insertion of a new smart card **100**, the reader **300** generates a session identifier. The session identifier is configured to identify a current session of a card insertion. The session identifier, for example, can be a pseudo- random number represented with two bytes of data or the session identifier can be a number that is incremented each time a smart card **100** is inserted and reset to zero when a predetermined value is reached. In this case, the reader **300** sends an INSERT message to the computer **700** or the set-top box **601**, which includes a card identifier as previously described above and a session identifier which is generated for each new smart card **100** insertion. All subsequent PRESS, RELEASE and MOVE messages need not include the card identifier but will include the session identifier and user interface element object data or press coordinates previously described.

[**0220**] When using a session identifier, the system **600** operates as described above with reference to FIGS. **6**(*a*) and **6**(*b*), except that the event manager **901**, upon receiving an INSERT message from the reader **300**, stores the session identifier as the current session identifier and a card identifier as the current card identifier. When the event manager **901** receives a PRESS, RELEASE or MOVE message, the event manager **901** checks that the session identifier is equal to the current session identifier. If so, the event manager **901** sets a card identifier used in all messages to the current card identifier. Otherwise, if the session identifier is not equal to the current session identifier, the event manager **901** informs the user, via the display manager **306**, and the display device **101**, that a message has been received without a corresponding INSERT message. The user, for example, is then requested to remove and reinsert the card **100**.

[**0221**] **1.7.2** Data Formats

[**0222**] The data format of the reader **300** protocol used in the system **600** is a fixed size header followed by a variable length data field which can be zero bytes or more in length, followed by an eight bit check-sum and complement.

[**0223**] 1.7.3 Message Header

[**0224**] The message header is preferably of a fixed length and is prepended to (i.e. appended to, but in front of) all messages sent from the reader **300** to a set top box **601** for example. It is necessary to keep the message header as small as possible due to any bandwidth restrictions that may be imposed. Table 9 below shows the format of the message header that is sent from a reader **300** to a remote module such as the set top-box **601** for example. The service and service-specific identifier are the same for a given smart card **100**. A service specific identifier is preferably set by a vendor for use with their application. The reader identifier (ID) of Table 9 is also in the header of each message. The reader identifier can be used by an application **304** to distinguish different users, for example, in a multi-player game application.

TABLE 9

Field	Description (Message Header Format)	Bytes
Preamble	Preamble to the message. Value is always 0xAA 0x55 (bit sequence 10101010 01010101). This is to make it easier for the event manager 901 to find the beginning of a message.	2
Version	The version of the user interface card infrared message protocol this messages uses. This version of the protocol is version 1(0x01 in the version field)	1
Type	Type of message. This is one of the values given in Table 10.	1
Reader ID	The 16 bit id of the reader that sent the message. This number is a pseudorandom generated number that is changed when the battery is replaced in the reader. This is needed to distinguish readers when multiple readers are being used with applications.	2
Service	Service identifier as stored on the card.	5
Service-specific	Service-specific identifier as stored on the card.	3

[**0225**] Table 10 shows a table listing the message event types that have been described above.

TABLE 10

Name	Description (Message Type Codes)	Code
INSERT	A card has been inserted into the reader.	'I'
REMOVE	The card has been removed from the reader.	'E'
PRESS	The touch panel has been pressed.	'P'
RELEASE	The press on the touch panel has been released.	'R'
MOVE	The press position has moved but the press has not been released.	'M'
BADCARD	A card has been inserted however it has not passed validation.	'B'
LOW_BATT	The battery in the reader is getting flat.	'L'

[**0226**] 1.7.3.1 Simple Messages

[**0227**] A number of message types are considered simple in that they consist solely of the message header described above followed by the message checksum byte and its complement. For example, a BADCARD message, a LOW_BATT message and a REMOVE message are simple messages. Table 11 shows the format of a simple message.

TABLE 11

Field	Description (Simple Message Format)	Bytes
Header	Message header as defined in Table 9.	14
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

[**0228**] **1.7.3.2** MOVE Messages

[**0229**] MOVE messages are formed of the message header described above followed by two fields defining the coordinates of the touch position on the touch panel **8** of the reader **300**. Table 12 shows the format of a MOVE message.

TABLE 12

Field	Description (Move Message Format)	Bytes
Header	Message header as defined in Table 9.	14
X	The X co-ordinate of the touch position.	1
Y	The Y co-ordinate of the touch position.	1
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

[0230] 1.7.3.3 PRESS and RELEASE Messages

[0231] Table 13 below shows the format of PRESS and RELEASE messages. PRESS and RELEASE messages, like MOVE messages contain the message header and touch coordinates. In addition, PRESS and RELEASE messages send data associated with a user-interface element if the touch position matches a user-interface element object defined on the smart card **100**. This data is of variable length, the actual size being defined by a corresponding smart card **100**. If the touched position does not match a user- interface element object defined on the smart card **100** (including if no user-interface elements are defined on the smart card **100**), zero bytes of data associated with user interface elements are sent. If there is no smart card **100** in the reader **300** then the service identifiers are all set to zero (ie OxOO) and zero bytes of data associated with the user-interface elements are transmitted to the remote module. The data associated with the user interface element normally corresponds to the data associated with the user interface element defined on the smart card **100** but may be modified or generated by processing on the smart card **100** or reader **300**.

TABLE 13

Field	Description (Press and Release Message Format)	Bytes
Header	Message header as defined by Table 9.	14
X	The X co-ordinate of the touch position.	1
Y	The Y co-ordinate of the touch position.	1
Length	The number of bytes of data. Can be zero.	2
Data	The data associated with the user interface element.	Length
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

[0232] 1.7.7 INSERT Messages

[0233] INSERT messages are formed of the message header described above and the contents of the card data object from an inserted smart card **100**. Table 14 below shows the format of an INSERT message.

TABLE 14

Field	Description (INSERT Message Format)	Bytes
Header	Message header as defined in Table 9.	14
Length	The number of bytes of data. Can be zero.	2
Data	The data from a Card Data object on the card.	Length
Checksum	Message checksum. This is the sum of all the bytes in the message.	1
Checksum'	The 1's complement of the checksum.	1

[0234] **FIG. 13** is a data flow diagram showing the flow of the above-described messages within the system **600** for a smart card **100**. As seen in **FIG. 27**, the card header **1100** and object structure **1213** are read by the CPU **1045** of the reader **300** which sends a corresponding INSERT, REMOVE, PRESS, RELEASE, MOVE, BADCARD or LOW BAT message to the event manager **901** via the I/O daemon **920**. The event manager **901** can have up to twenty-one core messages, which are sent to and received from the master launcher **902**, launcher **903** and applications **904**.

[0235] 1.8 Example

[0236] The operation of the system **600** will now be further explained with reference to the following example. The system **600** is customisable by virtue of a user being able to utilise a number of different smart cards **100** to perform corresponding different operations. For example, with particular reference to the system **600**B, **FIG. 14**(a) shows a memory card **100**C which according to the user interface elements **1414** printed thereon is configured for the retrieval of on-line music associated with an Internet site entitled "Blues Guitar Masters". The on-line music can be accessed over the system **600**B using the memory card **100**C and then purchased using a CPU card **100**D configured for use with an electronic banking application, as will be explained below with reference to FIGS. **15**(*a*) and **15**(*b*). Alternatively, other functions may be performed on the system **600**B, using different smart cards **100**, such as home shopping, ordering home delivery fast food such a pizzas, and the like. In each instance, insertion of an appropriate smart card **100** into the reader **300** causes a corresponding computer application to commence operation, either within the set-top box **601** or the computer system **700**, in order to service user commands entered via the reader **300** and to return appropriate information for audio- visual feedback to the user.

[0237] For the memory card **100**C, on-line music is provided as data to the set-top box **601** which permits reproduction of audio and any related visual images on the output device **616** or the display **701** of the computer system **700**. The user interface elements **1414** of the memory card **100**C are in the form of a "play button"**1401**, a "rewind button"**1403**, a "fast forward button"**1405**, a "stop button"**1407**, a "select button"**1409**, a "record button"**1411** and a two-way directional controller **1413**, printed on a front face **1416** of the memory card **100**C.

[0238] **FIG. 14**(*b*) is a table showing user interface element objects (e.g. **1431**) associated with each of the user interface elements **1401** to **1417**. The user interface element objects (e.g. **1431**) are stored in a memory (not shown) formed within the memory card **100**C similar to the memory **219** of the memory card **100**A. As described above with reference to Table **6** and as shown in **FIG. 14**(*b*), each of the user interface element objects (e.g. **1431**) has six fields being a flags field **1420**, an X1 field **1421**, a Y1 field **1422**, an X2 field **1423**, a Y2 field **1424** and a data field **1425**, describing the position of and data associated with a corresponding user interface element. For example, the flags field **1420** for the select button 1409 is a one byte field set to a hexadecimal value of '0×020' (0x representing hexadecimal notation), indicating that data associated with the select button **1409** is not to be sent in a release event, as described above with reference to Table 7. The X1 field **1421** associated with the select button **1409** is a one byte field set to a value of '0007' indicating the coordinate value of the bottom left hand point

1431 of the user interface element **1409** with respect to the top left point **1433** of the memory card **100**C, as described above with reference to Table 6. The data field **1425** is a variable size field which in the case of the select button **1409** is a value corresponding to an 'Enter' and/or 'Carriage Return' function.

[**0239**] The memory card **100**C also includes a card data object as described above with reference to Table 5. The card data object contains data that relates specifically to a particular smart card **100** and is normally sent as part of an INSERT message, upon the smart card **100** being inserted into the reader **300**. In the case of the memory card **100**C, the card data object indicates a URL, 'www.bluesguitarmasters.com', corresponding to the address of the 'Blues Guitar Masters' Internet site. A person skilled in the relevant art would appreciate that the URL is stored in the memory of the memory card **100**C in a digital format corresponding to the American Standard Code for Information Interchange (ASCII) values of the characters making up the URL. Alternatively, a card identifier can be stored in the memory of the memory card **100**C and can be mapped to the URL by the directory service **911**.

[**0240**] The memory card **100**C also includes a card identifier, as described with reference to Table 1, stored in the memory of the memory card **100**C. The card identifier includes a service identifier which in the case of the memory card **100**C can be mapped to the URL, 'www.bluesguitarmasters.com', corresponding to the address of the 'Blues Guitar Masters' Internet site by the directory service **911**. The card identifier also includes a service-specific identifier which in this case is a three-byte vendor identifier related to the vendor of the Blues Guitar Masters Internet site. The service-specific identifier can be assigned by the provider of the service (e.g. the vendor of the Blues Guitar Masters Internet site), and can be equal to any particular three-byte value. Each card **100** associated with the 'Blues Guitar Masters' Internet site, for example, can have a different service-specific identifier.

[**0241**] For the CPU card **100**D, the user interface elements **1514** are in the form of a numerical keypad **1560**, an "OK button"**1562**, a "cancel button"**1564**, a "clear button"**1566**, a "backspace button"**1568**, and a four way directional controller **1570** printed on the front face **1556** thereof.

[**0242**] Similar to the memory card **100**C, each of the user interface elements **1514** of the CPU card **100**D has at least one associated user interface element object (e.g. **1531**) stored in a storage means (not shown) formed within the CPU card **100**D similar to the storage means **276** of the CPU card **100**B. Again, as described above with reference to Table 6 and as shown in **FIG. 15**(b), each of the user interface element objects (e.g. **1531**) has six fields being a flags field **1520**, an X1 field **1521**, a Y1 field **1522**, an X2 field **1523**, a Y2 field **1524** and a data field **1525**, describing the position of and data associated with a corresponding user interface element. For example, the X1 field **1521** associated with the "number 2 button"**1540** of the numerical keypad **1560**, is a one byte field set to a value of "0005" indicating the coordinate value of the bottom left hand point **1541** of the user interface element **1540** with respect to the top left point **1533** of the CPU card **100**D. The data field **1525** of the "number 2 button"**1540** is a variable size field which in the case of the CPU card **100**D value corresponds to a hexadecimal representation corresponding to the ASCII value of the character '2'.

[**0243**] The CPU card **100**D also includes a card data object, as described above with reference to Table 5. In the case of the CPU card **100**D, the card data object indicates a URL (e.g. www.anz.com), corresponding to the address of an electronic banking Internet page. A person skilled in the relevant art would appreciate that the URL is stored in the storage means of the CPU card **100**D in a digital format corresponding to the ASCII values of the characters making up the URL.

[**0244**] Similar to the memory card **100**C, the CPU card **100**D also includes a card identifier, as described with reference to a Table I stored in the storage means of the CPU card **100**D. The card identifier includes a service identifier, which in the case of the CPU card **100**D can be mapped to the URL corresponding to the electronic banking application (e.g. www.anz.com). The card identifier also includes a service-specific identifier, which in this case is a three-byte vendor identifier. The service-specific identifier can be related to the vendor of the electronic banking package (e.g. the Australia New Zealand Banking Group Limited). Again, each CPU card **100**B associated with the electronic banking package, for example, can have a different service-specific identifier.

[**0245**] **FIG. 16** is a flow diagram showing the steps **1600** performed by a user in order to retrieve on-line music associated with the Blues Guitar Masters Internet site, over the card interface system **600**B. At the initial step **1601**, the user inserts the memory card **100**C into the reader **300**. As will be explained in further detail later in this document, having inserted the memory card **100**C into the reader **300**, an application associated with the Blues Guitar Masters Internet page commences, for example on the server computer **650**, and returns to the set-top box **601** for display on the output device **616** a first menu screen **1720**, as seen in **FIG. 17**(*a*), relating to a function to be performed, in this case a selection of Blues Guitar Masters. Then at the next step **1603**, using the reader **300** to select the two-way directional controller **1413**, the user scrolls through the various offerings to make a desired selection, in this case for an artist called Young Dead Guy. In response to the user's selection, a further menu screen **1722**, as seen in **FIG. 17**(*b*), is then displayed on the output device **616** advising the user of the possible selections that may be made. At the next step **1605**, the user scrolls the menu screen **1722** by selecting the two-way directional controller **1413**, and makes a further desired selection. In response to the user's selection at step **1605**, a further menu screen **1724** is then displayed on the output device **616** as seen in **FIG. 17**(*c*) advising the user of the further possible selections that may be made. In accordance with this example, the user can access a free sample video clip or a full concert music video corresponding to the selection at step **1605**, depending on the 'Quality of Service' that the user wishes to access from the Blues Guitar Masters Internet site. For example, if the user only wishes to access a one-minute free sample of the video clip corresponding to the selection at step **1605**, then the user can select the 'Free Sample' item **1726** on the menu screen **1724**. Otherwise, the user can select the 'Full Concert Video' item **1728** which will require the user to pay some associated fee for service. At the next step **1607**, the user selects the 'Full Concert

Video' item **1728**. In response to the user's selection at step **1607**, a further screen **1730** is then displayed on the output device **616** as seen in **FIG. 17**(*d*) advising the user to insert a payment card (e.g. the CPU card **100**D) into the reader **300**. The screen **1730** also advises the user of the fee amount and of a 'Biller Code' associated with the vendor of the Blues Guitar Masters Internet site. At the next step **1609**, the user removes the memory card **100**C and inserts the CPU card **100**D into the reader **300**. In accordance with the present example, at the next step **1611**, the user presses one or more user interface elements **154** representing a personal identification number (PIN) by applying pressure to the touch panel **308** on or adjacent to each of the associated user interface elements **154**. As will be explained in further detail later in this document, in response to the user entering the personal identification number, the CPU card **100** D can buffer the personal identification number in an input buffer and compare the buffered personal identification number to a personal identification number stored in a storage means (not shown) of the CPU card **100**D. If the CPU card **100**D determines that the entered personal identification number is identical to the stored personal identification number then an application associated with the banking application Internet site commences, for example on the server computer **652**, and returns to the set-top box **601** for display on the output device **616** a conventional banking application menu screen relating to a function to be performed, in this case a selection enabling the payment of the specified amount to the vendor identified by the displayed biller code. At the next step **1613**, the user utilises the user interface elements **1514** of the CPU card **100**D to operate the banking application in order to complete payment for the full concert music video of the young dead guy. Once payment is completed, the Blues Guitar Masters application then retrieves the selection (i.e. the full concert music video of the Young Dead Guy), which is then streamed to the set-top box **616** for appropriate output as seen in **FIG. 17**(*e*). Since the music video is, in effect, a series of "live" images, as compared to the substantially static images of the menu screens **1720**, **17221724** and **1730**, the music video may advantageously be obtained and/or streamed from another location on the computer network **720** not associated with the generation of the menu screens **1720**, **1722**, **1724** and **1730**.

[**0246**] **FIG. 18** is a flow diagram showing a method **1800** performed by the reader **300** for checking the memory card **100**A upon the user inserting the memory card **100**A into the reader **300** at step **1601** of the process **1600**. The method **1800** is also performed by the reader **300** at step **1609** in order to check the CPU card **100**D. The method **1800** checks for changes in the presence and validity of a smart card in the remote reader **300** and responds accordingly. The method **1800** is performed by the CPU **1045** and begins at step **1801** where if the memory card **100**C is inserted into the reader **300**, then the method **1800** proceeds to step **1802**. At step **1802**, if the inserted memory card **100**A is a new card (i.e. in the previous state there was no smart card **100** in the reader **300**), then the method **1800** proceeds to step **1803**. Otherwise, the method **1800** concludes. At the next step **1803**, "magic number" and "checksum" fields are read from a card header stored in the memory of the card **100**A by the CPU **1045**, and are checked for correctness. If the "magic number" and "checksum" are correct, then the method **1800** proceeds to step **1804**. In the case of the CPU card **100**B, the CPU **1045** only reads the magic number field as the reader **300** is not able to read the data stored on the CPU card **100**B. The method **1800** continues at step **1804**, where the card identifier is read from the card header and "Move events", "Don't Report Co-ordinates" and "Beep" flags are set by the CPU **1045**. The card data associated with the memory card **100**C is also read from the memory card **100**C at this step **1804**. At the next step **1805**, an INSERT message, including the card data, is sent to set top box **601** by the CPU **1045**, and the INSERT message is processed by the CPU **805**. Then at step **1806**, a "BEEP" is sounded and the method **1800** concludes.

[**0247**] If the "magic number" and "checksum" fields are not correct (ie: the memory card **100**C is not valid) at step **1803**, then the method **1800** proceeds to step **1810** where the don't beep, no move events and event co-ordinate flags are set. At the next step **1811**, a BAD CARD message is sent to the set top box **601** by the CPU **1045**, and the BAD CARD message is processed by the CPU **805** resulting in display of such a message on the output device **616**. Then at step **1812**, a "BOOP" is sounded and the method **1800** concludes.

[**0248**] If the memory card **100**C is not inserted into the reader **300** at step **1801**, then the method **1800** proceeds to step **1807**. At step **1807**, if this is the first operation of the reader **300** after the reset then the method **1800** concludes. Otherwise, the method **1800** proceeds to step **1808** where the "Beep", "No MoveEvents" and "No Event Co-ordinates" flags are set to default values by the CPU **1045** and the card identifier stored in memory of the memory card **100**C is set to "NO_CARD". At the next step **1809**, a REMOVE message is sent to the set top box **601** by the CPU **1045**, and the REMOVE message is processed by the CPU **805**. The method **1800** concludes after step **1809**.

[**0249**] **FIG. 19** is a flow diagram showing a method **1900** of scanning the touch panel **308** of the reader **300** of the system **600**B incorporating the software architecture **900**. The method **1900** is typically executed by the CPU **1045** at steps **1603**, **1605** and **1607** of the process of **FIG. 16** as the user scrolls the menu screens (e.g. **1720**) by selecting the user interface elements **1514**. The method **1900** checks for touch panel touches that equate with user interface element **1514** selections and responds accordingly. The method begins at step **1901** where if the touch panel **308** is being touched, then the method **1900** proceeds to step **1902**. Otherwise, the method **1900** proceeds to step **1912**, where if the touch panel **308** has been touched previously then the method **1900** proceeds to step **1913**. Otherwise, the method **1900** concludes.

[**0250**] At step **1913**, the "beep", "move events" and "no event co-ordinate" flags are set according to their corresponding values in the associated card header if a card is present. If a card is not present in the reader **300** then these flags are set to default values. Then at step **1914**, the message type is set to RELEASE and the method **1900** proceeds to step **1905**.

[**0251**] The method **1900** continues at step **802**, where if this is the first time that a touch has been noticed since there was no touch, then the method **1900** proceeds to step **1903**. At the next step **1903**, the CPU **1045** determines if a bad card has been inserted into the reader **300** by checking the result of step **1803**, then in the case that a bad card has been inserted into the reader **300**, the method **1900** proceeds to step **1915**. Then at step **1915**, a BAD Card message is sent

to the set top box **601**, the BAD CARD message is stored in memory **806**, and the method **1900** concludes. If it was determined at step **1903** that the memory card **100**C was valid, by checking the result of step **703**, or that no card was inserted into the reader **300**, by the checking of step **1801**, then the method **1900** proceeds to step **1904**, where the type of message is set to PRESS in the message header of Table **9**. At the next step **1905**, the CPU **1045** determines the touch coordinates (i.e. X, Y coordinates of user press location) via the touch panel interface **1041**. Then at the next step **1906**, the offset and scale functions are applied to the coordinates by the CPU **1045**. The offset and scale functions map the coordinate space of the touch panel **308** to the coordinate space of the card **100**C. The method **1900** continues at the next step **1907**, where if the CPU **1045** determines that the sent message was a MOVE and/or no card was inserted into the reader **300**, by checking step **1801**, then the method **1900** proceeds directly to step **1909**. Otherwise, the method **1900** proceeds to step **1908** and the memory of the memory card **100**C is searched by the CPU **1045** in order to find the first user interface element whose X1, Y1, X2, Y2 values form a range within which the touch coordinates fall and data associated with matched user interface element is read from the memory card **100**A. At the step **1909**, the message is sent along with any data to the set top box **601**, and the CPU **805** processes the message. The method **1900** continues at the next step **1911**, where a BEEP sound is sounded and the method **1900** concludes.

[**0252**] If this is not the first time that a touch has been noticed since there was no touch, at step **1902**, then the method **1900** proceeds to step **1916**. At step **1916**, if the touch detected at step **1901** was a move, then the method **1900** proceeds to step **1917**. Otherwise the method **1900** concludes. At step **1917**, the message type is set to MOVE and the method **1900** proceeds to step **1905**. For example, a MOVE message can be sent along with the X, Y coordinates of a touch position as defined by Tables, 12 and 15, a PRESS and RELEASE message can be sent along with X, Y coordinates of a touch position and data associated with a user interface object as defined by Tables, 12 and 16. If it was determined at step **1907** that the message was a MOVE, at step **1909**, then the CPU **1045** sends a MOVE message to the set top box **601**. The CPU **805** of the set top box **601** processes the X, Y coordinates as cursor information resulting in a cursor move that is displayed on the output device **616**. In this case, the next RELEASE message can be interpreted as a command to select the displayed object at the cursor position (e.g. to select an icon **1716** of the menu screen **1724**). Further, if NO Event Coordinates (see Table 2) have been set in the memory card **100**C, then the reader **300** may send the data associated with a user interface object to the event manager **301** in the computer **700** or set top box **601** without sending the X, Y coordinates of the touch position.

[**0253**] The method **1900** can be similarly performed at steps **1611** and **1613** for the CPU card **100**D. However, in the case of the CPU card **100**D, the mapping of the offset and scale functions to the coordinate space of the touch panel **308** of the card **100**D at step **1906** and the searching of the CPU card **100**D at step **1908**, is performed by the CPU (not shown) of the CPU card **100**D. The operation of the CPU cards **100**B and **100**D in this regard will be described in more detail later in this document.

[**0254**] **FIG. 20** is a flow diagram showing a process **2000** of events performed by the system **600**B incorporating the software architecture **900**. The process **2000**, is executed by the CPU **705**, depending on the configuration of the system **600**, upon the user inserting the memory card **100**C into the reader **300** at step **1601**. The process **2000** begins at step **2001** where a system initialisation process is performed to start the event manager **901**. At step **2001** the I/O daemon **920** is typically also started along with the event manager **901**.

[**0255**] At the next step **2002**, the event manager **901** starts the launcher **903**. Then at the step **2003**, the event manager **901** passes a message to the launcher **903**, enabling the launcher **903** to determine which application **904** to execute, and the launcher **903** then starts the corresponding application **904**. In the case of the memory card **100**C, the corresponding application **904** is an application associated with the Blues Guitar Masters Internet site. The process **2000** continues at the next step **2004**, where once the currently running application **904** is no longer needed (e.g. whilst the Blues Guitar Masters application selection is being streamed to the set-top box **616** from another location on the computer network **720** not associated with the generation of the menu screens **1720**, **1722**, **1724** and **1730** or when the CPU card **100**D is inserted into the reader **300** at step **1609**), the launcher **903** provides an exit message to the running application (i.e. the application associated with the Blues Guitar Masters Internet site) in order to end the execution of the running application. All applications are terminated when the system **600**B is powered down or switched off.

[**0256**] **FIG. 21** is a flow diagram showing a method **2100** of receiving an event performed by the event manager **901**, at step **2001**. The method **2100** can be executed by the CPU **705** for computer implementations. Alternatively, the method **2100** can be executed by the CPU **805** in set top box implementations. The method **2100** begins at step **2101**, where the launcher **903** is started. At the next step **2103**, the event manager **901** receives an event. If the event received at step **2103** is not from the reader **300** at the next step **2105**, then the method **2100** proceeds to step **2107** where a component identifier (XID) associated with the received event is checked and corrected if necessary. The method **2100** continues at the next step **2109**, where if the new application sending an event is allowed to send the event, then the method **2100** proceeds to step **2111**. At step **2111**, the event is sent to a destination process component and the method **2100** returns to step **2103**. If the sending application is not allowed to send the event at step **2109**, then the method **2100** proceeds to step **2113**, where the event is dropped and the method **2100** returns to step **2103**.

[**0257**] If the event is from the reader **300** at step **2105**, then the method **2100** proceeds to step **2115**. If the event is a BADCARD, LOWBAT, INSERT or REMOVE event at step **2115** then the method **2100** proceeds to step **2117**. Otherwise the method **2100** proceeds to step **2119**. At step **2117**, the event is passed to the launcher **903** and the method **2100** returns to step **2103**. If the card identifier is the NO_CARD identifier at step **2119**, then the corresponding message is passed to the launcher **903** at step **2117**. Otherwise the method **2100** proceeds to step **2121**, where the service identifier portion of the card identifier is compared with the service identifier used in determining the current front application (i.e. the application to receive messages

from the event manager). If the service identifier is not the same as that which has been used to determine the front application, then the method **2100** proceeds to step **2117** where this message is passed to the launcher **903**. Otherwise, the method **2100** proceeds to step **2123**, where the event is sent to the front application and the method **2100** returns to step **2103**.

[**0258**] **FIG. 22** is a flow diagram, showing an overview of the method **2200** performed by the launcher **903** during steps **2002** to **2004** of the process **2000**. The method **3700** can be executed by the CPU **705** for computer implementations. Alternatively, the method **2200** can be executed by the CPU **4305** in set top box implementations or by the CPU of a remote server. The method **2200** begins at the first step **2201**, where the launcher **903** connects to the event manager **901**, and then continues to a next step **2202** where persistent applications (i.e. applications such as browser controllers that will generally always be running while the system **600** is operating) are started. At the next step **2203**, the launcher **903** waits for an event and when an event is received the launcher **903** proceeds to step **2205**. If the event is the NO_CARD identifier at step **2205**, then the process proceeds to step **2207**. Otherwise the method **2200** proceeds to step **2209**. At step **2207**, the launcher **903** performs a predetermined system specific function (e.g. displays a message on the output device **616**) in response to the NO_CARD identifier and the method **2200** returns to step **2203**.

[**0259**] If the event at decision step **2205** is determined not to be a NO_CARD identifier, another decision step **2209** is entered to determine whether or not the event is a PRESS, RELEASE, REMOVE, MOVE or INSERT. If this decision step **2209** returns a "yes", that is, the event is one of the aforementioned events, then the method **2200** proceeds to step **2227**. Otherwise the method **2200** proceeds to a further decision step **2213**. At step **2227**, the launcher **903** changes the application and the method **2200** returns to step **2203**. A REMOVE event followed by an INSERT event would occur at step **1609** of the process **1600** when the user removes the memory card **100**C from the reader **300** and inserts the CPU card **100**D. The INSERT event results in the reader **300** sending an INSERT message containing the URL of the banking application to the CPU **805** of the set top box **601** which passes the URL to a server (e.g. the server computer **652**) associated with the banking application.

[**0260**] If the event at step **2209** is not one of the PRESS, RELEASE, REMOVE, MOVE or INSERT events, then a decision step **2213** is entered. This decision step **2213** makes a determination on a BADCARD or LOW_BATT event. If the event is a BADCARD or LOW_BATT event at step **2213**, then the method **2200** proceeds to step **2215**, otherwise the method **2200** proceeds to step **2217**. At step **2215**, the launcher **903** gives the user feedback on the event that has occurred (e.g. displaying a "Low Battery" message on the output device **616** if the LOW_BATT event is determined or a "Incorrect Card" upon determination of a BADCARD event) and the method **2200** returns to step **2203**. If the event at decision step **2213** is neither a BADCARD or LOW_BATT event, then step **2217** is entered.

[**0261**] If the event is an APP_REGISTER event at step **2217**, then the method **2200** proceeds to step **2229**, application registering. Otherwise the method **2200** proceeds to step **2225**. At step **2229**, the application is registered (i.e. the application informs the other components **901**, **902** and **906** that it is now ready to receive messages) and the method **2200** returns to step **2203**. At step **2225**, the event is discarded and the method **2200** returns to step **2203**.

[**0262**] 2.0 CPU CARD OPERATION

[**0263**] The operation of the CPU cards described above will be discussed below with reference to the CPU card **100**B. However, a person skilled in the relevant art would appreciate that the following description applies equally to the CPU card **100**D.

[**0264**] The CPU card **100**B has several advantages over the memory card **100**A and over other conventional memory card or user interface card platforms. Firstly, a personal identification number (PIN) and other data may be kept secure by ensuring that all coordinate translation from the reader **300**, in regard to a user selecting one or more of the user interface elements **154**, is executed by the CPU **259** of the CPU card **100**B, thus providing some protection against discovery and copying of the data stored on the card **100**B. In order to provide such security, the user interface element objects (see Table 5) are stored in the storage means **276** of the CPU card **100**B. The CPU card **100**B can protect the data stored in the storage means **276** by read/write protecting the storage means **276**. Secondly, data returned by the CPU card **100**B can be the result of a transformation executed by the CPU **275** and applied to a certain sequence of user interface elements **154** being selected. The CPU card **100**B can itself store the sequence in the storage means **276** and release the necessary data only after the card **100**B has detected that a valid sequence has been selected. For example, entry of a personal identification number by a user can be used to authenticate a user of the card **100**B, prior to which other features provided by the CPU card **100**B are locked out.

[**0265**] A third advantage of the CPU card **100**B over the memory card **100**A, and over other conventional memory cards and user interface card platforms, is that the CPU card **100**B can offer secure connection between the CPU card **100**B and a remote application executing on the computer system **600** or remote server computers **650**, **652**. The CPU card **100**B can be configured with cryptographic functions together with a Public Key Infrastructure (PKI) to establish a secure communications channel between the CPU card **100**B and a remote application, even when working with an unsecured reader **300**.

[**0266**] Still another advantage of the CPU card **100**B is that the CPU card **100**B can be loaded with several different application programs, which can interact with one another. For example, card resident applications, such as secure online payment applications, can provide additional functionality to the standard user interface card application described below, allowing data generated by these applications to be associated with user interface elements.

[**0267**] 2.1 Communications

[**0268**] The CPU card **100**B preferably communicates with the reader **300** using the "T=0" protocol as defined in the "International Standards Organisation (ISO)/International Electrotechnical Commission (IEC) 7816-3: 1997 Standards, Part 3: Electronic Signals and Transmission Protocol", [hereinafter referred to as the 'ISO 7816-3 standards']. However, any other suitable bidirectional protocol (e.g. T=l)

can be used for communications between the CPU card **100**B and the reader **300**. As will be described in detail below, the reader **300** issues different commands to the CPU card **100**B depending on a user interface element **154** selected by the user. All commands issued to the CPU card **100**B by the reader **300** are in an 'Application Protocol Data Unit (APDU)" format as described in the "ISO/IEC 7816-4 Standards, Part 4: Electronic Signals and Transmission Protocol"[hereinafter referred to as the 'ISO 7816-4 standards']. The format and fields of the commands issued to the CPU card **100**B will be described in detail below.

[**0269**] 2.2 Operating System

[**0270**] The CPU card **100**B is configured to utilise any generic multi-application operating platform such as the 'Multos' operating system or 'JavaCard'. However, the CPU card **100**B can also utilise a single application operating platform.

[**0271**] 2.3 Operating Procedure

[**0272**] **FIG. 23** is a flow diagram showing a method **2300** of operating the CPU card **100**B using the reader **300**. The method **2300** is implemented as software, such as an application program executing within the CPU **1045** of the reader **300** and being stored in the memory **1046**. In particular, the steps of the method **2300** are effected by instructions in the software that are carried out by the CPU **1045**. The underlined text in the flow diagram of **FIG. 23** indicate steps of the method **2300** which require a command (typically invoked by the reader **300**) to be issued to software stored in the storage means 276 of the CPU card **100**B and being executed by the CPU **275**. For example, at step **2313** the CPU **1045** of the reader **300** sends an (X, Y) position coordinate pair associated with a user interface element **154** selection to the CPU **275** of the CPU card **100**B for processing, using a PROCESS COORD command, as will be explained in more detail below. As described above, commands are configured as application protocol data units (APDU) having a format as described in the ISO 7816-4 standard. However, any other suitable existing or custom protocol can be used.

[**0273**] The software executing on the CPU card **100**B may be divided into two separate parts being an operating system and an application program. The operating system (hereinafter referred to as the 'Card Operating System') is configured to manage the interface between an 'application' program and the reader **300**, where the card operating system forms the platform for the execution of applications on the CPU card **100**B. The application program provides user interface functions of the CPU card **100**B. The dashed line **2302** in **FIG. 23** indicates steps (i.e. steps **2305** to **2317**) in which the application program is active. The application program will be hereinafter referred to as the 'User Interface Card Resident Application'.

[**0274**] As described above, user interface element objects as described herein, represent mapping data, which relate the user interface elements **154** directly imprinted on a surface of the CPU smart card **100**B, to commands or addresses (eg: Uniform Resource Locators (URLs)). Mapping data includes (X,Y) coordinate pairs which typically define the size and location of user interface elements **154** on the CPU card **100**B, as described above for the memory card **100**A. The user interface element objects are stored in the storage means **276** of the CPU card **100**B and are not able to be accessed by the reader **300**.

[**0275**] The method **2300** begins at step **2301** where the CPU card **100**B is powered up. The CPU card **100**B can be powered up automatically upon being inserted into the reader **300** by a user or after a user touches the touch panel **308**. At the next step **2303**, the user interface card resident application, stored in the storage means **276**, is selected by the operating system executing within the CPU **275** upon the operating system receiving a command from the CPU **1045** of the reader **300**. According to the ISO 7816-4 standard, CPU card application programs are selected by a SELECT FILE command. The format of the SELECT FILE command sent by the CPU **1045** of the reader **300** to the CPU card **100**B is shown in Table 15 below.

TABLE 15

Field	Value
CLA (Class)	0x00
INS (Instruction)	0xA4
P1 (Parameter 1)	0x04
P2 (Parameter 2)	0x0C
Lc (Command Data Length)	Length of user interface card resident application identifier
Data (Command Data)	User interface card resident application identifier
Le (Response Data Length)	0

[**0276**] The SELECT FILE command can result in further commands sent by the reader **300** being re-directed to the user interface card resident application rather than the operating system. The SELECT FILE command will fail if the user interface card resident application does not exist in the storage means **276** on the CPU card **100**B or cannot be selected for another reason. The process performed by the CPU card **100**B in response to the SELECT FILE command will be explained in further detail below with reference to **FIG. 42**. A 'user interface card resident application identifier', referred to in Table 15, is an identifier associated with the user interface card resident application. The user interface card resident application identifier is configured in accordance to the guidelines set out in "International Standards Organisation (ISO)/International Electrotechnical Commission (IEC) 7816-5: 1994 Standards, Part 5: Numbering System and Registration Procedure for Application Identifiers"[hereinafter referred to as the 'ISO 7816-5 standards']. The user interface card resident application identifier consists of a registered application provider identifier (RID) assigned by a standards authority, and a proprietary application provider extension (PIX) assigned by the application provider.

[**0277**] The method **2300** continues at the next step **2305** where the CPU **1045** of the reader **300** reads a card header **1100** associated with the card **100**B. The card header **1100** was described above with reference to **FIG. 11** and is stored in the storage means **276** of the CPU card **100**B. Step **705** is implemented using a READ BINARY command, according to the ISO 7816-4 standard, sent by the CPU 1045. If the user interface card resident application is implemented in accordance with an ISO 7816-4 standard file system, as will be explained below, then the card header **1100** is contained in an elementary file (EF) having an identifier 0x000. Such an elementary file is implicitly selected when the user interface card resident application is selected at step **2303**. The format of the READ BINARY command invoked by the reader **300** is shown in Table 16 below.

TABLE 16

Field	Value
CLA	0x00
INS	0xB0
P1	0x00
P2	0x00
Lc	0
Data	Empty
Le	0x13

[0278] As explained above, the card header **1100** is 19 bytes long and if any error or warning identifiers are encountered at step **2305**, then the reader **300** will assume that the card header **1100** cannot be read properly and register a bad card. The process performed by the CPU card **100**B in response to the READ BINARY command will be explained in further detail below with reference to **FIG. 43**. At the next step **2307**, if the reader **300** is in low-power mode, then the method **2300** proceeds to step **2311**. Otherwise, at the next step **2309**, the state of the smart card **100**B is restored as will be explained in detail below. Step **2309** is implemented using a RESTORE STATE command sent to the CPU **275** by the CPU **1045** of the reader **300** and which is configured as an application protocol data unit having a format as shown in Table 17 below.

TABLE 17

Field	Value
CLA	0x00
INS	0x12
P1	0x00 (other values are RFU)
P2	0x00 (other values are RFU)
Lc	Length of state code
Data	State code
Le	0

[0279] After receiving the RESTORE STATE command with the parameters shown in Table 17 from the CPU **1045** of the reader **300**, the user interface card resident application executing within the CPU **275** checks that a present state code provided by the user interface resident card application is identical to a previous state code generated during a previous SAVE STATE command stored in non-volatile memory of the storage means **276** of the CPU card **100**B and in the memory **1046** of the reader **300**. The previous state code is provided to the user interface card resident application in the 'Data' field of the RESTORE STATE command. The SAVE STATE command will be explained in further detail later in this document. If the present state code is not identical to the previous state code generated during a previous SAVE STATE command then previous state information stored in non-volatile memory of the storage means **276** is erased. Warning and error conditions shown in Table 18 below can also occur. However, if the present state code is identical to the previous state code received from the reader **300**, then current volatile state information is replaced with the state information stored in the non-volatile memory of the storage means **276**. The process performed by the CPU card **100**B in response to the RESTORE STATE command will be explained in further detail below with reference to **FIG. 41**.

TABLE 18

Value	Meaning
0x6300	Invalid code presented—state erased
0x6B00	Incorrect parameters P1–P2

[0280] At step **2311**, the CPU **1045** waits for an event such as the selection of a user interface element **154** in the form of a press or release. Upon selection of a user interface element **154** by the user, the method **2300** proceeds to step **2313** where the CPU **1045** of the reader **300** sends an (X, Y) position coordinate pair associated with the selection to the CPU **275** of the CPU card **100**B for processing. Step **2313** is implemented using a PROCESS COORD command configured as an application protocol data unit sent to the CPU card **100**B by the CPU **1045** of the reader **300** and having a format shown in Table 19 below.

TABLE 19

Field	Value
CLA	0x90
INS	0x00 (PRESS) or 0x02 (RELEASE)
P1	X co-ordinate
P2	Y co-ordinate
Lc	0
Data	Empty
Le	Maximum length of data to receive

[0281] As will be explained in further detail later in this document, the user interface card resident application processes the coordinate data by comparing the (X, Y) position coordinate pair to the data stored in the storage means **276** of the microprocessor **259**. The reader **300** typically sets the maximum length of data to receive (Le) field to a maximum possible value, OxOO (which is used to encode ×100). Data associated with a user interface element object, which is stored in the storage means **276** of the microprocessor **259** and which is associated with the (X, Y) position coordinate pair sent from the reader **300**, is returned by the user interface card resident application in response to the PROCESS COORD command. The format of the data returned by the user interface card resident application in response to the PROCESS COORD command is shown in Table 20 below. The process performed by the CPU card **100**B in response to the PROCESS COORD command will be explained in further detail below with reference to **FIG. 28**.

TABLE 20

Name	Length	Description
flags	1 byte	Flags specific to a matched User Interface Element
data	0 or more bytes	Data associated with the matched User Interface Element

[0282] Certain user interface element objects may be designated to be active only on a press or a release event corresponding to a user selection, and separate instruction codes can be used to allow the reader **300** to distinguish one from the other. If the (X, Y) position coordinate pair supplied by the reader **300** is in the range [(0,0), (127, 255)]

but no data is found to match the position coordinate data, then the "flags" field, as shown in table 20, is constructed from global button flags specified in the card header **1100**. Further, the data field is returned empty and one of the error identifiers shown in Table 21 below may also be returned by the user interface card resident application.

TABLE 21

Value	Meaning
0×6985	Data must be encrypted, but no session key available, or session key invalid
0×6A82	There is no user interface card image loaded (File not found)
0×6A86	The co-ordinates provided are outside the allowable range (Incorrect P1–P2)

[**0283**] If the CPU **1045** has not received any events for a predetermined period (e.g. one minute), at step **2311**, then the method **2300** proceeds to step **2315** where the user interface card resident application executing within the CPU **275** saves current state information in non-volatile memory of the storage means **276**. Step **2315** is implemented using a SAVE STATE command sent from the reader **300** and which is configured as an application protocol data unit having a format shown in Table 22 below.

TABLE 22

Field	Value
CLA	0×90
INS	0×10
P1	0×00
P2	0×00
Lc	0
Data	Empty
Le	Maximum length of random data to be generated

[**0284**] After receiving the SAVE STATE command with correct parameters the user interface card resident application executing within the CPU **275** saves all volatile state information to non-volatile memory of the storage means **276** and then generates a random state code of a specified length. The user interface card resident application also returns the state code to the reader **300**. The state code is saved in the memory **1046** of the reader **300** in order to use the RESTORE STATE command described above. The process performed by the CPU card **100**B in response to the SAVE STATE command will be explained in further detail below with reference to **FIG. 40**. One of the error identifiers shown in Table 23 below may also be returned to reader **300** by the user interface application upon receiving the SAVE STATE command.

TABLE 23

Value	Meaning
0×6300	State saved, but no random number returned
0×6B00	Incorrect parameters P1–P2
0×6CXX	Incorrect Le specified. XX specifies recommended length.

[**0285**] The error identifier 0×6CXX shown in Table 23 is returned to the reader **300** by the user interface card resident application executing within the CPU **275** when the CPU card **100**B is unwilling to accept the requested state code length because it is too short. The user interface card resident application can also provide a value corresponding to the shortest appropriate length along with the error condition. Alternatively, if the requested length is longer than the CPU card **100**B can handle, then the user interface card resident application generates the longest code that the CPU card **100**B is configured for and returns a value corresponding to the longest code. A typical length for the state code is between eight and sixteen bytes. However, the state code can be of any length up to two hundred and fifty six bytes.

[**0286**] The method **2300** continues at the next step **2317**, where the CPU card **100**B is powered down by the operating system executing in the CPU **275**. Then after a predetermined period, the CPU **1045** of the reader **300** assigns the reader **100**B to low power mode at the next step **2319**. At the next step **2321**, the reader **300** is woken up and the method **2300** then returns to step **2301**. The reader **300** can be woken up automatically by the CPU **1045**, for example, upon a CPU card **100**B being inserted into the reader **300** as shown in **FIG. 4** and/or by a user touching the touch panel **308**.

[**0287**] 2.3.1 CPU Card Operating Modes

[**0288**] The user interface card resident application executing within the CPU **275** of the CPU card **100**B operates in one of two different operating modes (i.e. Standard Input Mode and Buffered Input Mode), which affect the response of the user interface card resident application to a touch on the touch panel **308**.

[**0289**] 2.3.1.1 Standard Input Mode

[**0290**] Standard input mode is the default mode. In standard input mode, the CPU **275** of the CPU card 1 OOB returns the data field associated with a given user interface element **154** if a match is found in the storage means **276**, in response to a PROCESS COORD command sent from the reader **300**. Conversely, if a match is not found, the CPU **275** returns default data. The returned information may be inline data (i.e. stored within the definition of the particular user interface element), file data (i.e. stored within a file in a file system implemented on the CPU card **100**B), data indicating a transition to buffered input mode or data returned by a delegated application, as will be explained in more detail below.

[**0291**] 2.3.1.2 Buffered Input Mode

[**0292**] When the user interface card resident application is in buffered input mode, the PROCESS COORD command does not result in the return of any data to the reader **300** in response to the selection of a user interface element **154**. Instead, a default flags field is returned by the CPU **275** to the CPU **1045** of the reader **300** in response to the selection of a user interface element **154**. Certain user interface elements are designated as input user interface elements, which, when pressed, cause an associated object identifier to be stored in a temporary input buffer.

[**0293**] Additionally, four special user interface elements may be defined. One of these user interface elements **154** is the "OK button"**162**, which serves to commit the temporary input buffer and perform some action on the buffered data. The other special user interface elements are the "cancel button"**254**, the "backspace button"**258** and the "clear button"**256**, which represent "Cancel", "Backspace" and "Clear" roles, respectively. A user interface element **154**

configured as a "Cancel" button, when selected, clears an input buffer of the CPU card **110**B and returns the user interface card resident application to standard mode, in exactly the same state as the user interface card resident application was in when buffered input mode was activated. A user interface element configured as a "Backspace" button, when selected, clears a reference to a last entered user interface element from an active buffer of the CPU card **100**B. If the buffer is empty, no action is taken. Finally, a user interface element **154** configured as a "Clear" button, when selected, removes references to all user interface elements **154** contained within the input buffer of the CPU card **110**B. If the buffer is already empty, no action is taken.

[**0294**] The properties of a particular buffered input session are associated with a buffer descriptor, which is read from data stored on the CPU card **100**B as will be described later in this document. The buffer descriptor has the structure shown in Table 24 below:

TABLE 24

Name	Length	Description
Flags	1 byte	Various parameters about the buffer descriptor. See Table 25.
Action	1 byte	A code, indicating the action to be performed after the OK button is pressed. See Table 26.
Butlen	1 byte	The maximum number of bytes that can be stored in the buffer.
1_buttons	1 byte	The length of the subsequent "buttons" field. Must be equal to 4 or more bytes, in order to accommodate the special user interface element definitions.
Buttons	1_buttons	An array storing the identifiers of the user interface elements that can be used as buffer elements. See Table 27.
1_data	1 byte	The length of the subsequent "data" field. Its value depends on the specified action.
Data	1_data	Other data specific to the requested action.
Output	Variable	If the buffered input mode is invoked by pressing a user interface element on the card, this data is sent to the reader as data when that user interface element is pressed. This field may be empty.

[**0295**] **FIG. 49**(*a*) shows an example buffer descriptor **4900** which has the configuration described in Table 24. The buffer descriptor **4900** includes an array **4901**, referred to as the 'Buttons' array, which is used to store identifiers associated with user interface elements **154** that can be used as buffer user interface elements. The array **4901** of **FIG. 49**(*a*) includes three identifiers **4902** corresponding to user interface elements designated as input user interface elements. The array **4901** also includes the four special user interface elements **4903** (i.e. OK, Cancel, Clear, Bksp) described above. Each of the identifiers stored in the buffer descriptor **4900** can have associated mapping data **4905**, as shown in **FIG. 49**(a). The buffer descriptor **4900** can be utilised to configure the function of the user interface elements of a CPU card. For example, **FIG. 49**(b) shows a CPU card **100**F, which has been configured for a pizza ordering application. The CPU card **100**F comprises a user interface element **4907**, which includes the text 'Start Over'. The Start Over user interface element **4907** has been configured as a 'Clear' button (having the function described above) by storing an associated identifier, '5', in the clear identifier position of the array **4901**. The identifier '5' indicates the position of a corresponding user interface element object **4911** in the mapping data **4905**.

[**0296**] The flags for the flags field of the buffer descriptor structure of Table 24 are described in Table 25 below:

TABLE 25

Name	Value	Description
Circular	0×01	In a circular buffer, a character that overflows the buffer replaces the oldest character. If this flag is not set, overflowing characters are ignored.
expand data	0×02	Normally, buffer data is stored and processed as the Object identifiers corresponding to the user interface elements that were pressed. If this flag is set, the buffer data is expanded to the contents of the user interface elements that were pressed before being processed.

[**0297**] The actions described in Table 26 below are defined for the action field of the structure of Table 24:

TABLE 26

Name	Value	Description
Output	0×00	An output buffer.
Passcode	0×10	A passcode buffer.

[**0298**] The "buttons" field has the format shown in Table 27 below:

TABLE 27

Name	Length	Description
Ok_id	1 byte	The identifier of the "OK button".
Cancel_id	1 byte	The identifier of the "Cancel button".
Bksp_id	1 byte	The identifier of the "Backspace button".
Clear_id	1 byte	The identifier of the "Clear button".
Input_id	Variable	An array storing the identifiers of the user interface elements that can be used as input elements.

[**0299**] The "ok_id" field shown in Table 27 stores a valid object identifier. The other three special user interface elements do not need to be specified and can be set to a default identifier (e.g. defined by the value 0).

[**0300**] The user interface card resident application executing within the CPU **275** of the CPU card **100**B can be configured to begin in buffered input mode when the application is initialised. This is enabled by setting up an appropriate buffer descriptor in a file with a certain identifier (i.e.0×0002), as an "initial" buffer descriptor (as described above in Table 28). A typical use of this feature is to disable all of the features of the CPU card **100**B until a pass-code (e.g. a personal identification number) is entered. As an example, the pass-code can be in the form of a series of user interface element objects associated with user interface elements **154** associated with an image (e.g. a human face) formed on a surface of a CPU card. For example, **FIG. 48** shows a CPU card **100**E depicting an image **4801** of a face. In this example, the card **100**E includes ten regions (e.g.

4803), shown in phantom lines which define user interface elements arbitrarily positioned on the surface of the user interface card **100**E. However, in this case the user interface elements defined by the regions (e.g. **4803**) are not visible to a user of the card **100**E. The user's pass-code can be made up, for example, of the regions **4804**, **4805** and **4806** adjacent the left-eye, the left-side of the mouth and the right-eye, respectively, of the image **4801**, and the user can enter the pass code by selecting these regions **4804**, **4805** and **4806** in sequence.

[**0301**] 2.3.1.3 Output Buffers

[**0302**] When a buffer descriptor is configured to have an action field of "output", as shown in Table 26, an associated buffer is referred to as the output buffer. When a user interface element (e.g. user interface element **162**) configured as an "OK button" (i.e. having the function described above in section 2.3.1.2) is selected, a response is created by appending buffered data to what is stored in the "data" field of the buffer descriptor associated with the user interface element **162** and stored in the storage means **276**. If the "expand data" flag is set, the buffered object identifiers are replaced with the data associated with the user interface element that the object identifiers represent.

[**0303**] When a user interface element (e.g. the user interface element **164**) configured as a "Cancel button" (i.e. having the function described above in section 2.3.1.2) is selected, only the header **1100** is returned. The user interface card resident application can be configured to enter standard input mode after the user interface elements configured as "OK" (e.g. **162**) and configured as "Cancel" (e.g. **164**) are selected.

[**0304**] 2.3.1.4 Pass-code buffers

[**0305**] When a buffer descriptor is configured to have an action field of "passcode", as shown in Table 26, the associated buffer is a Pass-code buffer. The "data" field of the buffer descriptor as shown in Table 24 stores the identifier of a file containing the required pass-code for a particular CPU card **100**B. When the user interface element **162** configured as an "OK" button is selected, a string stored in a temporary input buffer in the storage means **276** is compared against a string stored in the pass-code file corresponding to the identifier. A standard message can be configured to indicate whether or not a correct pass-code was entered by a user. The CPU card **100**B exits buffered input mode only if a pass-code match occurs.

[**0306**] For security, a cancel function should not be defined in a pass-code buffer descriptor. The pass-code buffer descriptor is preferably specified as an initial buffer.

[**0307**] 2.4 Additional Software Interfaces

[**0308**] 2.4.1 International Standards Organisation (ISO) File System

[**0309**] An international standards organisation (ISO) file system can be implemented on the CPU card **100**B to provide users of the CPU card **100**B with access to certain information stored in the storage means **276** of the CPU card **100**B. Such a file system preferably has a flat directory structure containing transparent elementary files (EF) referenced by 16-bit identifiers in accordance with ISO 7816-4 standards. However, a person skilled in the relevant art would realise that any suitable file system and corresponding directory structure can be used on the CPU card **100**B.

[**0310**] Table 28 below lists elementary files and associated identifiers, which can be configured on the CPU card **100**B.

TABLE 28

File ID	File Name	File Description
0×0000	User interface card header	The header portion of the user interface card
0×0001	User interface card data	The data portion of the user interface card
0×0002	Initial buffer descriptor	A buffer descriptor indicating that the card should be initially in buffered input mode
0×0008	Card key	A cryptographic key used by the CPU card (for example, to facilitate secure transmission of a new session key)
0×0009	Session key	A previously negotiated cryptographic key used to encrypt data destined for the remote application
0×0101	SL1 key	A cryptographic key used to authenticate access to security level 1
0×1102	SL2 key	A key used to authenticate access to security level 2
0×1103	SL3 key	A key used to authenticate access to security level 3

[**0311**] A 'User Interface Card Header' file having identifier (0×0000), as shown in Table 28, is implicitly selected when the user interface card resident application executing on the CPU card **100**B is activated (e.g. at step **2303** of the method **2300**). The data in the User Interface Card Header file would typically conform to the format **1100** as shown in. **FIG. 11**.

[**0312**] The data associated with a single file is stored in one contiguous area of non-volatile memory in the storage means **276** of the CPU card **100**B. Each file is assigned a maximum length, and the data region associated with that file is preferably configured to provide enough space to store a longest possible file.

[**0313**] A file directory can be used to store certain information about each file present in the file system where the file directory consists of a list of entries beginning at address 0×10 of a static data area in the storage means **276**. A two-byte word at address OxOO can be used to represent the total number of files currently present in the directory. The region between 0×02 and 0×10 of the static data area is configured as an array of pointers to the To directory entries for files 0×0000 through 0×0006, providing for quicker access to these particular files. Each directory entry is 25 bytes long, and consists of the following fields as shown in Table 29 below:

TABLE 29

Name	Length	Description
Id	2 bytes	The file's two-byte ISO 7816 identifier
Flags	2 bytes	Flags associated with the file
Offset	2 bytes	The offset of the first byte of the file from the bottom of the static data area
Read level	1 byte	The minimum security level allowed to read the file
Write level	1 byte	The minimum security level allowed to write the file

TABLE 29-continued

Name	Length	Description
Length	2 bytes	The current length of the file
Maxlen	2 bytes	The maximum possible length of the file
RFU	4 bytes	Currently unused - reserved for future use

[0314] The flags associated with the file are a combination of zero or more of the following values shown in Table 30:

TABLE 30

Value	Name	Description
0×80	erasable	Parts of the file may be erased if write permissions satisfied
0×40	hide length	Do not report the length of the file when selected, unless read permissions are already satisfied

[0315] The elementary files of the ISO file system can be accessed through the following commands.

[0316] 2.4.1.1 File Control Information

[0317] When selecting a file stored in the storage means **276** of the card **100**B, the user can receive a structure containing file control information. This structure preferably complies with Table 2 of the ISO 7816-4 standard, which includes the following:

[0318] (i) File length—this can be suppressed by the CPU **275** if a security level of the CPU card **100**B is lower than a corresponding read level for the file, as will be explained in further detail later in this document;

[0319] (ii) File descriptor byte—as outlined in Table 3 of the ISO 7816-4 standard;

[0320] (iii) File identifier; and

[0321] (iv) File security attributes—can be configured as a two byte number where a first byte can describe a read security level of a file and a second byte can describe a write level for the file.

[0322] 2.4.1.2 Select File

[0323] The CPU card **100**B can be configured such that a user can select a file stored in the storage means **276** for viewing or editing. The file can be associated with one of the user interface elements **154** (e.g. the user interface element **162**). In this case, upon selecting the associated user interface element **162**, the CPU **1045** of the reader **300** sends a SELECT FILE command to the user interface card resident application executing within the CPU **275**. The SELECT FILE command is configured as an application protocol data unit and has a format as shown in Table 31 below. The SELECT FILE command is a subset of the specification for such a command as outlined in the ISO 7816-4 standards.

TABLE 31

Field	Value
CLA	0×00
INS	0×A4

TABLE 31-continued

Field	Value
P1	0×00 (select MF, DF, or EF) 0×02 (select EF under current DF) Both 0×00 and 0×02 produce equivalent results
P2	0×00 (select first or only occurrence, return File Control Information (FCI) 0×0C (select first or only occurrence, return no data)
Lc	2
Data	two-byte file identifier
Le	maximum length of FCI buffer (or 0, if no FCI requested)

[0324] If the two-byte file identifier as shown in Table 31 points to a valid file, then that file becomes an active file. File control information (FCI) is returned to the reader **300** by the user interface card resident application if the P2 parameter is set to '0×00'. The error and warning identifiers as seen in Table 32 below can also be returned by the user interface card resident application upon the user interface card resident application receiving a SELECT FILE command.

TABLE 32

Value	Meaning
0×6283	File exists, but empty
0×6A81	Incorrect parameters P1–P2
0×6A82	File not found
0×6A87	Lc incorrect

[0325] 2.4.1.3 Read Binary

[0326] Up to two hundred and fifty six bytes can be read from a selected elementary file stored in the storage means **276** upon the reader **300** sending a READ BINARY command to the user interface card resident application executing within the CPU **275**. If the end of a file is reached or exceeded then all available bytes of the file are read and a warning code is returned to the reader **300** by the user interface card resident application. The READ BINARY command is configured as an application protocol data unit and has a format as shown in Table 33 below.

TABLE 33

Field	Value
CLA	0×00
INS	0×B0
P1	MSB of offset, must be <0×80 since short EF addressing not allowed
P2	LSB of offset
Lc	0
Data	Empty
Le	Number of bytes to be read

[0327] The error and warning identifiers as seen in Table 34 below can also be returned to the CPU **1045** of the reader **300** by the user interface card resident application upon the user interface card resident application receiving a READ BINARY command.

TABLE 34

Value	Meaning
0×6282	End of file reached
0×6982	File's read permissions are not satisfied by current security level
0×6A81	Incorrect parameters (P1>=0×80)
0×6B00	offset beyond end of file

[0328] 2.4.1.4 Write Binary and Update Binary

[0329] WRITE BINARY and UPDATE BINARY commands are used to write one or more bytes to an elementary file stored in the storage means **276** of the CPU card **100**B. The WRITE BINARY and UPDATE BINARY commands are configured as application protocol data units having a format as shown in Table 35 below.

TABLE 35

Field	Value
CLA	0×00
INS	0×D0 (WRITE) or 0×D6 (UPDATE)
P1	MSB of offset, must be <0×80 since short EF addressing not allowed
P2	LSB of offset
Lc	Number of bytes to write
Data	Bytes to write
Le	0

[0330] Up to two hundred and fifty-five bytes of data can be written to a selected elementary file stored in the storage means **276** using the WRITE BINARY and UPDATE BINARY commands. When writing to file offsets where data is already present, that data is replaced with the new data specified in the WRITE BINARY and UPDATE BINARY commands. If the end of a file is exceeded, the length of the file is extended to accommodate the entire data string being written by the WRITE BINARY and UPDATE BINARY commands, unless the maximum length specified in the properties of the file being written to is exceeded.

[0331] The error and warning identifiers as seen in Table 36 below can be returned to the CPU **1045** of the reader **300** by the user interface card resident application upon the user interface card resident application receiving a WRITE BINARY and UPDATE BINARY command.

TABLE 36

Value	Meaning
0×6381	Maximum length exceeded - full length not written
0×6982	File's write permissions are not satisfied by current security level
0×6A81	Incorrect parameters (P1>=0×80)
0×6B00	Offset beyond end of file

[0332] 2.4.1.5 Erase Binary

[0333] An ERASE BINARY command can be sent to the user interface card resident application by the reader **300** to truncate a file stored in the storage means **276** of the CPU card **100**B, according to an offset indicated by the parameters of the ERASE BINARY command.

[0334] The format of the ERASE BINARY command is shown in Table 37 below:

TABLE 37

Field	Value
CLA	0×00
INS	0×0E
P1	MSB of offset, must be <0×80 since short EF addressing not allowed
P2	LSB of offset
Lc	0
Data	Empty
Le	0

[0335] The ERASE BINARY command truncates a file from the offset indicated by the parameters **P1** and **P2**. Further, the following error and warning identifiers can be returned by the user interface card resident application upon receiving an ERASE BINARY command, as shown in Table 38 below:

TABLE 38

Value	Meaning
0×6981	File is not erasable
0×6982	File's write permissions are not satisfied by current security level
0×6A81	Incorrect parameters (P1>=0×80)
0×6B00	Offset beyond end of file

[0336] 2.4.2 Security architecture

[0337] At any given time, the user interface card resident application is executing at a certain security level on the microprocessor **259** of the CPU card **100**B. Four security levels are described herein. However, up to thirty-two security levels can be implemented on the CPU card **100**B. The user interface card resident application executing within the CPU **275** is configured to begin at security level zero, as defined in Table 39 below:

TABLE 39

SL	User Type	Description
0	Default	Restricted access, normal operation within the reader.
1	Application	Used by the remote application to establish a session key through a trusted set-top box or reader.
2	Owner	Used by the user of the CPU card to customise data such as personal details, passcode etc within a file.
3	Issuer	Used by the card's issuer for administrative purposes.

[0338] To increase the security level of the CPU card **100**B the user follows the authentication process as will be described below with reference to **FIGS. 46 and 47**. In accordance with the described authentication process, the CPU **275** of the card **100**B provides random challenge data in response to a GET CHALLENGE command. The challenge data is provided by the CPU **275** for subsequent encryption and returned to the CPU **275** with an EXTERNAL AUTHENTICATE command as will be described below with reference to scetion 2.4.2.2.

[0339] The user interface card resident application decrypts the encrypted challenge data according to a key provided by a user and compares this with the original challenge. If the decrypted challenge matches the original challenge then a user is authenticated and the CPU card

100B is set to a required security level as requested by the user. Otherwise, the CPU **275** denies permission to increase the security level of the CPU card **100**B.

[**0340**] Generally, no authentication is required to decrease the security level of the CPU card **100**B.

[**0341**] Each file of a file system implemented on the CPU card **100**B is associated with a read security level, and a write security level. The READ BINARY, WRITE BINARY, UPDATE BINARY and ERASE BINARY commands sent to the CPU card **100**B from the reader **300** will only succeed if the current security level of the CPU card **100**B is greater than or equal to the appropriate security level associated with the selected file.

[**0342**] The security attributes of a file are encoded in file control information (FCI) as a two-byte field marked by a tag '**86**', which is associated with a particular file. The first byte represents the read security level, while the second byte represents the write security level associated with the file. The following commands are sent to the CPU **275** of the CPU card **100**B by the CPU **1045** of the reader **300** during the process of changing the security level of the CPU card **100**B.

[**0343**] The PROCESS COORD command can be utilised to effectively override the security level of a file. A file that is referenced in a user interface element object specified by the **P1** and **P2** co-ordinates of an PROCESS COORD command can be accessed without the card **100**B being set to the required security level, if a user interface element **154** corresponding with the file (i.e. the user interface element object) is pressed or released. The file can only be accessed in this manner through the mechanisms provided by the PROCESS COORD command. However, this enables the issuer of the card **100**B to provide limited access to the contents of a file while still protecting the file from general use.

[**0344**] 2.4.2.1 Get Challenge

[**0345**] The format of the GET CHALLENGE command follows a subset of a similar command specification in the ISO 7816-4 standard, as shown in Table 40.

TABLE 40

Field	Value
CLA	0x00
INS	0x84
P1	0x00
P2	0x00
Lc	0
Data	Empty
Le	Variable (must be positive multiple of 8)

[**0346**] A number of bytes (i.e., determined by the Le field) of challenge data are returned by the CPU **275** of the CPU card **100**B in response to the GET CHALLENGE command. Further, the error and warning identifiers as shown in Table 41 can also be returned by the CPU **275** in response to the GET CHALLENGE command.

TABLE 41

Value	Meaning
0x6A81	Incorrect parameters

[**0347**] 2.4.2.2 External Authenticate

[**0348**] If the security level requested by a user is greater than the security level that the CPU card **100**B is currently set to, then an EXTERNAL AUTHENTICATE command having a format shown in Table 42 below, is sent to the CPU **275**. The EXTERNAL AUTHENTICATE command attempts to decrypt the security level data stored in the storage means **276** using a key associated with the requested security level. If the EXTERNAL AUTHENTICATE command is successful, then the security level of the CPU card **100**B is changed in accordance with the request made by the user. Conversely, if the requested security level is less than or equal to the security level that the CPU card **100**B is currently set to then the security level is changed without attempting to authenticate the user. A nominal number of attempts (e.g. three) can be allowed before a fresh challenge is requested by the CPU **1045**.

TABLE 42

Field	Value
CLA	0x00
INS	0x82
P1	0x00
P2	0x80 & S, where S is a 5-bit number representing the requested SL
Lc	Length of key to encrypt challenge
Data	Encrypted challenge
Le	0

[**0349**] The error and warning identifiers shown in Table 43 can also result from an EXTERNAL AUTHENTICATE command being sent to the CPU card **100**B.

TABLE 43

Value	Meaning
0x63CX	Authentication failed: X more attempts are allowed before a new challenge must be issued
0x6700	Lc inconsistent with parameters (must be equal to 64 if authentication required)
0x6985	No challenge data available: GET CHALLENGE command must first be issued
0x6A81	Invalid parameters P1–P2
0x6A88	Key file for referenced security level not available

[**0350**] 2.4.2.3 Example

[**0351**] In one example of the security architecture described herein, the CPU card **100**B can contain some information about its user. The information can be associated with one or more user interface elements. For example, one user interface element object can store the user's home address, while another user interface element object stores the user's credit card number. If the issuer of the CPU card **100**B wishes to make the data associated with both user interface element objects easy to modify at a later date without having to change the mapping data associated with

the user interface element objects, both pieces of user information can be stored in separate files in the memory **276** of the card **100**B, and references to these files can stored in a mapping data file associated with the user interface element objects.

[**0352**] If the user's place of residence were to change, the user can be allowed to change the address by giving the associated file a write security level of 2, so that the user must authenticate themselves before being able to change the information. However, if the user desires to change the credit card number stored on the card **100**B, the issuer might require that the user bring the card **100**B to a central authority, for example, together with the new credit card, at which point the issuer can update the credit card number on the CPU card **100**B. Such a file can be created with a write security level of 3, denying the user the opportunity to modify the file, but allowing the issuer to do so.

[**0353**] 2.5 Instruction Invocation

[**0354**] The commands and classes referred to herein are configured in accordance with the ISO 7816-4 standard. Whenever a command configured as an application protocol data unit is received by the user interface card resident application executing within the CPU **275**, an instruction pointer is set to zero. The user interface card resident application analyses a header associated with the received application protocol data unit to determine whether a valid command has been received. If valid, a process corresponding to the command is executed by the CPU **275** of the CPU card **100**B. Alternatively, if the command is not valid then an error identifier (e.g. '0×6E00' representing 'class not supported') is returned to the CPU **1045** of the reader **300**.

[**0355**] 2.5.1 Main Process

[**0356**] A main process **2400**, as shown in **FIG. 24**, is executed by the CPU **275** upon a command being issued to the user interface card resident application by the reader **300**. The process **2400** checks whether or not the CPU card **100**B has been initialised and if not, then an initialisation process, as seen in **FIG. 25**, is executed by the CPU **275**. The initialisation process will be explained in detail below.

[**0357**] The main process **2400** is implemented as software, preferably as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process **2400** begins at step **2401** upon the user interface card resident application receiving a command. If an 'Initialised Flag' stored in the storage means **276** is set then the process **2400** continues at the next step **2403**. Otherwise, the process **2400** proceeds to step **2405**, where the CPU **275** executes the initialisation process **2500** in order to set the user interface card resident application to a standard or buffered input mode. At step **2403**, if the class of the command is '0×90' then the process **2400** proceeds to step **2407** where a proprietry instruction process is executed by the CPU **275**, on all commands not conforming to ISO 7816-4 standard interfaces but which are specific to the user interface card resident application. Otherwise the process **2400** proceeds to step **2409** where if the class of the command is '0×00' then the process **2400** proceeds to step **2411**. The proprietary instruction process executed at step **2407** will be explained in detail below with reference to **FIG. 26**. At step **2411**, an ISO instruction process is executed by the CPU **275** on all commands not conforming to ISO 7816-4 standard interfaces. The ISO instruction process will be explained in more detail below with reference to **FIG. 27**. If the class of the command is not '0×00', at step **2409**, then the process **2400** proceeds to step **2413**, where a 'bad class' error identifier is returned to the CPU **1045** of the reader **300** by the CPU **275** of the CPU card **100**B and the process **2400** concludes.

[**0358**] 2.5.2 Initialisation Process

[**0359**] **FIG. 25** is a flow diagram showing the initialisation process executed at step **2405** by the CPU **275**, at the beginning of the first instruction issued to the CPU card **100**B. The initialisation process is implemented as software, preferably as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of step **2405** begins at sub-step **2501**, where if there is a valid initial buffer descriptor described in a file stored in the storage means **276**, then the process of step **2405** proceeds to sub-step **2503**, where the buffered input mode is set according to the parameters of the initial buffer descriptor. If there is no valid initial buffer descriptor, at sub-step **2501**, then the process of step **2405** proceeds to sub-step **2505** where the CPU card **100**B is set to standard input mode. At the next sub-step **2505**, the initialised flag stored in the storage means **276** is set and the process by the CPU **275** of step **2405** concludes.

[**0360**] 2.5.3 Proprietary Instruction Process

[**0361**] **FIG. 26** is a flow diagram showing the proprietary instruction process executed at step **2407**. The process of step **2407** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As described above, the proprietary instruction process is responsible for all of the commands which do not conform to the ISO 7816-4 standard interfaces, but are specific to the user interface card resident application and have a class byte, 0×90.

[**0362**] The process of step **2407** begins at sub-step **2601**, where if the CPU **275** determines that the command received by the user interface card resident application is 0×12 representing a RESTORE STATE command then the process proceeds to sub-step **2603**. Otherwise, the process of step **2407** proceeds to sub-step **2605** where if the CPU **275** has saved the state information in the storage means **276** then the process proceeds to sub-step **2607**.

[**0363**] At sub-step **2603**, a restore state process is executed by the CPU **275**. The restore state process is configured to erase previous state information stored in non-volatile memory of the storage means **276** if the present state code is not identical to the previous state code generated during a previous SAVE STATE command. The restore state process will be described in more detail below with reference to **FIG. 41**.

[**0364**] At sub-step **2607**, the state information and state code are erased from the non- volatile memory of the storage means **276**. The process of step **2407** continues at the next sub-step **2609** where if the command received by the user interface card resident application is 0×10 representing a SAVE STATE command, then the process continues at sub-step **2611**. Otherwise, the process of step **2407** continues

at sub-step **2613**. At sub-step **2611**, a save state process is executed by the CPU **275**. The save state process saves all volatile state information to non-volatile memory of the storage means **276** and then generates a random state code of a specified length. The save state process will be described in more detail below with reference to **FIG. 40**.

[**0365**] At sub-step **2613**,if the command is 0×00 or 0×02 representing a PROCESS COORD coordinate command then the process of step **2407** proceeds to sub-step **2615** where a process coordinate process is executed by the CPU **275**. The process coordinate process searches the data stored on the storage means **276** to determine a user interface element object corresponding to the coordinates provided by the reader **300** upon a user selecting one of the user interface elements **154**. The process coordinate process will be described below with reference to **FIG. 28**. Otherwise, the process of step **2407** proceeds to sub-step **2617** where a bad instruction error identifier is returned to the CPU **1045** of the reader **300** by the user interface card resident application executing within the CPU **275**.

[**0366**] 2.5.4 ISO instruction process

[**0367**] **FIG. 27** is a flow diagram showing the ISO instruction process executed at step **2411** of the process **2400**. The process of step **2411** is responsible for processing all instructions, received by the user interface card resident application executing within the CPU **275**, from the reader **300**, which do not conform to the interfaces specified in the ISO 7816-4 standards, and have a class byte of 0×00. The process of step **2411** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of step **2411** begins at sub-step **2701** where if the command received by the CPU **275** of the CPU card **100**B is 0×A4, representing a SELECT FILE command then the process proceeds to sub-step **2703**. Otherwise, the process of step **2411** proceeds to sub-step **2705**. At sub-step **2703** a select file process is executed by the CPU **275** in order to determine an active file. The select file process will be described in more detail below with reference to **FIG. 42**.

[**0368**] At sub-step **2705**, if the command received by the CPU **275** of the CPU card **100**B is 033 B0, representing a READ BINARY command then the process proceeds to sub-step **2707**. Otherwise, the process of step **2411** proceeds to sub-step **2709**. At sub-step **2707** a read binary process is executed by the CPU **275** to read data from a currently selected elementary file. The read binary process will be described in more detail below with reference to **FIG. 43**.

[**0369**] The process of step **2411** continues at sub-step **2709**, where if the command received by the CPU **275** of the CPU card **100**B is 0×D0 or 0×D6, representing a WRITE BINARY command or an UPDATE BINARY command, respectively, then the process proceeds to sub-step **2711**. Otherwise, the process of step **2411** proceeds to sub-step **2713**. At sub-step **2711** a write binary process is executed to write data to a currently selected elementary file. The write binary process will be described in more detail below with reference to **FIG. 44**.

[**0370**] At sub-step **2713**, if the command received by the CPU **275** of the CPU card **100**B is 0×0E, representing an ERASE BINARY command then the process proceeds to sub-step **2715**. Otherwise, the process of step **2411** proceeds to sub-step **2717**. At sub-step **2715** an erase binary process is executed by the CPU **275** to truncate a currently selected elementary file depending on the parameters received with the ERASE BINARY command. The erase binary process will be described in more detail below with reference to **FIG. 45**.

[**0371**] The process of step **2411** continues at sub-step **2717**, where if the command received by user interface card resident application executing within the CPU **275** of the CPU card **100**B is 0×84, representing a GET CHALLENGE command then the process proceeds to sub-step **2719**. Otherwise, the process of step **2411** proceeds to sub-step **2721**. At sub-step **2719** a get challenge process is executed by the CPU **275** to provide challenge data in order to allow the security level of the CPU card **100**B to be changed. The get challenge process will be described in more detail below with reference to **FIG. 46**.

[**0372**] At sub-step **2721**, if the command received by the CPU **275** of the CPU card **100**B is 0×82, representing an EXTERNAL AUTHENTICATE command then the process proceeds to sub-step **2723**. Otherwise, the process of step **2411** proceeds to sub-step **2725**. At sub-step **2723** an external authenticate process is executed by the CPU **275** to decrypt presented challenge data using a key associated with a requested security level. The external authenticate process will be described in more detail below with reference to **FIG. 47**.

[**0373**] At sub-step **2725** a bad instruction error identifier is returned by the user interface card resident application executing within the CPU **275** and the process of step **2411** concludes.

[**0374**] 2.6 Process Coordinate Process

[**0375**] **FIG. 28** is a flow diagram showing a process coordinate process executed at sub- step **2615** of the process of **FIG. 26**. The process coordinate process is executed by the CPU **275** upon a PROCESS COORD command being received by the user interface card resident application executing within the CPU **275**, in response to the selection of a user interface element **154** by a user. The process coordinate process searches the data stored in the storage means **276** of the CPU card **100**B to determine a user interface element object corresponding to (X,Y) coordinates provided by the reader **300** in response to a touch on the touch panel **308**. The coordinates of the touch are contained in the PROCESS COORD command. The process of sub-step **2615** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of sub-step **2615** begins at sub-step **2801** where if the (X,Y) coordinates received from the CPU **1045** of the reader **300** are not within the range (127, 255), then the process proceeds to sub-step **2805**. Otherwise, the process of sub-step **2615** proceeds to sub-step **2803** where the CPU **275** searches the user interface element objects stored within the storage means **276**. If there are no user interface element objects left to search at sub-step **2803** then the process of sub-step **2615** proceeds to sub-step **2809**. Otherwise the process proceeds to sub-step **2807** where the flags and type of the next object searched, are determined by examining the fields of the object structure associated with the object. If the object is a user

interface element object and is active then the process of sub-step **2615** proceeds to sub-step **2813**. Otherwise, the process returns to sub-step **2803**. At sub- step **2813**, if the (X,Y) coordinates supplied by the CPU **1045** of the reader **300** as part of the PROCESS COORD are within the region specified for the particular user interface element object then the process proceeds to sub-step **2815**. Otherwise, the process of sub- step **2615** returns to sub-step **2803**. At sub-step **2815** a user interface handler process is executed by the CPU **275** to determine the data associated with the particular user interface element **154** selected by the user. The user interface element handler process will be described in more detail below with reference to **FIG. 29**.

[**0376**] The process of sub-step **2615** continues at the next sub-step **2817** where an output object data process is executed by the CPU **275**. The output object data process is executed when the CPU **275** has to output any data to the reader **300** and will be described in more detail below with reference to **FIG. 39**.

[**0377**] At sub-step **2809**, the output flags associated with a default user interface element are set and the process proceeds to sub-step **2817**.

[**0378**] The process of sub-step **2615** concludes at sub-step **2805** where an invalid (X,Y) error is returned to the CPU **1045** of the reader **300** by the user interface card resident application executing within the CPU **275**.

[**0379**] 2.6.1 User Interface Element Handler Process

[**0380**] The user interface element handler process of **FIG. 29** is executed at sub-step **2815** of the process coordinate process, in order to examine a user interface element object associated with a selected user interface element **154**. The process of sub-step **2815** determines what action the CPU **275** should take in response to the (X,Y) coordinates received by the user interface card resident application at step **2801**. The process of sub-step **2815** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of sub-step in **2815** begins at sub- step **2901** where if the CPU card **100**B is in buffered input mode then the process proceeds to sub-step **2903**. Otherwise, the process of sub-step **2815** proceeds to sub-step **2905**. At sub-step **2903** if the input event is a press event, then the process of sub-step **2815** proceeds to sub-step **2907** where a buffered input user interface element handler process is executed by the CPU **275**. Otherwise, the process of sub-step **2815** proceeds to sub-step **2911**. The buffered input user interface element handler process is executed to append a user interface element selection to an input buffer and will be described in more detail below with reference to **FIG. 30**.

[**0381**] At sub-step **2905**, if the input event is a press event (i.e. associated with a user interface element **154** selection), then the process of sub-step **2815** proceeds to sub-step **2909**. Otherwise, the process proceeds to sub-step **2913**. At sub-step **2909**, if a "no press" flag associated with the received command is set, then the process proceeds to sub- step **2911** where the output flags associated with a default user interface element are set and the process of sub-step **2815** concludes. At sub-step **2913** if a "no release" flag is set then the process proceeds to sub-step **2911**. Otherwise, the process of sub-step **2815** proceeds to sub-step **2915**.

[**0382**] The process of sub-step **2815** continues at sub-step **2915** where if the data associated with the selected user interface element **154** is inline data (as described above) then the process of step **2815** proceeds to sub-step **2919**. Otherwise, the data associated with the selected user interface element **154** is file data and the process proceeds to sub-step **2917**. At sub-step **2919**, the inline data associated with the selected user interface element **154** is read by the CPU **275**. At sub-step **2917**, an inline header associated with the data is stored in an output buffer. The process of sub-step **2815** continues at sub-step **2921** where the data associated with the selected user interface element **154** is read from an associated file stored in the storage means **276**. At the next sub-step **2923**, the user interface element object type associated with the selected user interface element **154** is determined. If the user interface element object is a delegator object then the process of sub-step **2815** proceeds to sub-step **2925** where a delegator user interface element handler process is executed by the CPU **275**. The delegator user interface element handler process will be described in more detail below with reference to **FIG. 38**. Alternatively, if the object associated with the selected user interface element is a buffer user interface element object, then the process of sub-step **2815** proceeds to sub-step **2927**. At sub-step **2927**, a buffer user interface element handler process, which will be explained below in more detail with reference to **FIG. 37**, is executed by the CPU **275**. Still further, if the object associated with the selected user interface element **154** is a text user interface element object, then the process of sub-step **2815** proceeds to sub-step **2929**. At sub-step **2929**, a text user interface element handler process, which will be explained below with reference to **FIG. 36**, is executed, by the CPU **275**, on the object data associated with the selected user interface element **154**. The process of step **2815** concludes after the user interface element handler process of sub-steps **2925**, **2927** and **2929** have been executed by the CPU **275**.

[**0383**] **2.6.2** Buffered Input User Interface Element Handler

[**0384**] **FIG. 30** is a flow diagram showing the buffered input user interface element handler process executed at sub-step **2907** of the process of **FIG. 29**. The process of sub-step **2907** is executed upon the selection of a user interface element **154** when the CPU card **100**B is in buffered input mode, in order to append the object data associated with the selected user interface element to an input buffer. The process of sub-step **2907** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of sub-step **2907** begins at sub-step **3001**, where a buffer descriptor associated with the current input buffer session is searched in order to determine if the buffer descriptor contains the object identifier of the user interface element corresponding to the (X,Y) coordinates, received at step **2801**. At the next sub-step **3003**, if the buffer descriptor designates the selected user interface element as an "OK button" (i.e. having the function described above) then the process of sub-step **2907** proceeds to sub-step **3005**. Otherwise, the process proceeds to sub-step **3007**. At sub- step **3005**, a buffer OK process is executed by the CPU **275**, which will be explained in more detail below with reference to **FIG. 31**. At sub-step **3007**, if the buffer descriptor designates the selected user interface element as a "cancel

button" (i.e. having the function described above) then the process of sub-step **2907** proceeds to sub-step **3009**. Otherwise, the process proceeds to sub-step **3011**. At sub-step **3009** a buffer cancel process is executed by the CPU **275**, which will be explained in more detail below with reference to **FIG. 33**.

[**0385**] The process of sub-step **2907** continues at sub-step **3011** where if the buffer descriptor designates the selected user interface element as a "clear button" (i.e. having the function described above) then the process proceeds to sub-step **3013**. Otherwise the process of sub-step **2907** proceeds to sub-step **3015**. At sub-step **3013** the input buffer is cleared by the CPU **275**. If the buffer descriptor designates the selected user interface element as a "backspace" button (i.e.having the function described above) at sub-step **3015** then the process proceeds to sub-step **3017** where a buffer backspace process is executed by the CPU **275**. The buffer backspace process will be explained in more detail below with reference to **FIG. 34**. If the buffer descriptor does not designate the selected user interface element as a "backspace" button at sub-step **3015** then the process proceeds to sub-step **3019**. At sub-step **3019**, if the selected user interface element is valid for the given input buffer then the process of sub-step **2907** proceeds to sub-step **3021**. Otherwise, the process proceeds to sub-step **3023** where the output flags associated with the default user interface element are set and the process of sub-step **2907** concludes. A buffer append process is executed by the CPU **275** at sub-step **3021**, which will be explained in more detail below with reference to **FIG. 35**.

[**0386**] Following the buffer OK and buffer cancel process executed at sub-steps **3005** and **3009** respectively, the process of sub-step **2907** proceeds to sub-step **3025** where the output flags associated with the buffer descriptor are set and the process concludes.

[**0387**] 2.6.3 Buffer OK Process

[**0388**] The buffer OK process of sub-step **3005**, as seen in **FIG. 31**, is executed by the CPU **275**, when the CPU card **100**B is in buffered input mode and a user interface element configured as an "OK button" (i.e. user interface element **162**) has been selected by a user at step **3003**. The process of sub-step **3005** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of sub-step **3005**, begins at sub-step **3101** where if an "expand data" flag of the buffer descriptor associated with the current input buffer is set, then the process proceeds to sub-step **3103**. Otherwise, the process proceeds to sub-step **3105**. At sub-step **3103** the data of the user interface element object associated with the selected user interface element is expanded into a working buffer. At sub-step **3105**, the working buffer is set as the current input buffer.

[**0389**] The process of sub-step **3005** continues at the next sub-step **3107** where if the buffer action field of the buffer descriptor is set to pass-code mode, then the process proceeds to sub-step **3109**. Otherwise, the process proceeds to sub-step **3111**. At sub- step **3109** a pass-code checker process is executed by the CPU **275** to determine whether there are valid pass-code entry attempts remaining and if so, whether the entered pass- code stored in the input buffer matches a predetermined pass-code stored in the storage means **276** of the CPU card **100**B. The pass-code checker process will be explained in more detail below with reference to **FIG. 32**. At sub-step **3111**, the data in the data field of the buffer descriptor is copied into an output data. The process of sub-step **3005** continues at the next sub-step **3113** where the contents of the working buffer are copied into the output buffer. At the next sub-step **3115**, the CPU card **100**B is set to standard input mode. The process of sub-step **3005** concludes at the next sub-step **3117** where the input buffer is cleared.

[**0390**] 2.6.4 Pass-code Checker Process

[**0391**] The pass-code checker process executed at sub-step **3109** of the process of **FIG. 31**, is invoked when user interface card resident application executing on CPU **275** of the CPU card **100**B is in buffered pass-code input mode and the user interface element **162** is selected. The process of sub-step **3109** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As, seen in **FIG. 32**, the process of sub-step **3109** begins at sub-step **3201** where if a pass-code retry counter stored in the storage means **276** is greater than zero then the process proceeds to sub-step **3203**. Otherwise, the process of sub-step **3109** proceeds to sub-step **3215** where the output buffer is set to "wrong password". At sub-step **3203**, a pass-code file referenced in the buffer descriptor data is read by the CPU **275**. If the pass-code matches the input buffer at the next sub-step **3205**, then the process proceeds to sub-step **3207** where an initial value is written to the pass-code retry counter. Otherwise, the process of sub-step **3109** proceeds to sub-step **3209** where the pass-code retry counter is decremented and the process proceeds to sub-step **3215**. After sub-step **3207** the process of sub-step **3109** proceeds to sub-step **3211** where the output buffer is set to "pass-code OK". The process of sub-step **3109** concludes at sub-step **3213** where the user interface card resident application executing on the CPU card **100**B is set to standard input mode.

[**0392**] 2.6.5 Buffer Cancel Process

[**0393**] The buffer cancel process as executed at sub-step **3009** during the process of **FIG. 30** is invoked when the CPU card **100**B is in buffered input mode and the user interface element **164** configured as a "cancel button" is selected by a user. The buffer cancel process results in the cancellation of buffered input mode (e.g. pass-code entry). However, in an implementation where pass-code entry is mandatory, a "cancel button" is not defined in the active buffer descriptor. The process of sub-step **3009** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As seen in **FIG. 33**, the process of sub-step **3009** begins at sub-step **3301** where if the buffer action is set to pass-code mode as defined by the action field of the buffer descriptor for the input buffer, then the process proceeds to sub-step **3303**. Otherwise, the process proceeds to sub-step **3304** where the data defined by the data field of the current buffer descriptor is copied into the output buffer. At sub-step **3303** the output buffer is set to "pass-code cancel". The process of sub-step **3009** continues at the next sub-step **3305** where the CPU card **100**B is set to standard input mode. The process concludes at the next sub-step **3307** when the input buffer is cleared.

[0394] 2.6.6 Buffer Backspace Process

[0395] The buffer backspace process as executed at sub-step **3017** of **FIG. 30** and is invoked when the CPU card **100**B is in buffered input mode and a user interface element **154**, which is configured as a backspace button (i.e. user interface element **168**) is selected by a user. The process of sub-step **3017** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As seen in **FIG. 34**, the process of sub-step **3017** begins at sub-step **3401** where if the input buffer length is greater than zero, then the process of sub-step **3017** proceeds to sub-step **3403**. Otherwise, the process of sub-step **3017** concludes. At sub-step **3403**, the last byte in the input buffer is cleared. The process of sub-step **3017** concludes at the next sub-step **3405** where the input buffer length is decremented.

[0396] 2.6.7 Buffer Append Process

[0397] The buffer append process as executed at sub-step **3021** of **FIG. 30** is invoked when the CPU card **100**B is in buffered input mode and a valid user interface element **154** is selected by a user. The process of sub-step **3021** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As seen in **FIG. 35**, the process of sub-step **3021** begins at sub-step **3501** where if the length of the input buffer is less than a maximum length then the process proceeds to sub-step **3503**. Otherwise, the process proceeds to sub-step **3505**. At sub-step **3503** the length of the input buffer is incremented. The process of sub-step **3021** continues at the next sub-step **3507** where the last byte in the input buffer is set to the current object identifier corresponding to the data object associated with the selected user interface element. At sub-step **3505** if the input buffer is circular, then the process proceeds to sub-step **3509**. Otherwise, the process concludes. At sub-step **3509** the bytes in the input buffer are moved one place to the left and process proceeds to sub-step **3507**. The process of sub-step **3021** concludes after sub-step **3507**.

[0398] 2.6.8 Text User Interface Element Process

[0399] The text user interface element process of sub-step **2929**, is executed to process a standard user interface element **154** having associated data which is stored in a corresponding user interface element object definition stored in the storage means **276**. The process of sub-step **2929** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As seen in **FIG. 36**, the process begins at sub-step **3601** where the flags associated with the selected user interface element corresponding to the (X,Y) coordinates received at step **2801**, are set for output by the CPU **275**. At the next sub-step **3603** if data is associated with the selected user interface element **154**, then the process proceeds to sub-step **3605**. Otherwise, the process of sub- step **2929** concludes. At sub-step **3605** the data associated with the selected user interface element **154** is appended to the output buffer and the process of sub-step **2929** concludes.

[0400] 5 2.6.9 Buffered User Interface Element Process

[0401] The buffered user interface element process, as executed at sub-step **2927** of **FIG. 29**, checks a buffer descriptor associated with the current input buffer session and enters the CPU card **100**B into buffered input mode. The process of sub-step **2927** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**.

[0402] As seen in **FIG. 37**, the process of sub-step **2927** begins at sub-step **3701** where the input buffer descriptor is examined by the CPU **275** to determine whether the buffer descriptor is valid and whether data is present in the data field of the buffer descriptor. If the result of sub-step **3701** is true, then the process of sub-step **2927** proceeds to sub-step **3703** where the user interface resident application executing on the CPU card **100**B is set to buffered input mode using the data associated with the user interface element **154** selected at sub-step **2801**. This data is used as the buffer descriptor. At the next sub-step **3707**, the output field of the buffer descriptor is appended to the output buffer and the process of sub-step **2927** concludes. If the result of sub-step **3701** is false, then the flags associated with the selected user interface element **154** are set for output and the process of sub-step **2927** concludes.

[0403] 2.6.10 Delegator User Interface Element Process

[0404] The delegator user interface element process of sub-step **2925** reads the object data associated with the user interface element **154** corresponding to the (X,Y) coordinates received by the user interface card resident application and determines an application identifier and an application protocol data unit to be sent to the application referenced by the identifier (i.e. the delegated application). The process of sub-step **2925** then invokes the identified application and returns the result of the application as the data associated with the selected user interface element. The 'Multos' delegate command can be used to invoke the identified application. Therefore, as discussed above, the CPU card **100**B can be loaded with several different application programs, which can interact with one another. This allows card resident applications (e.g secure online payment applications), to provide additional functionality to the user interface card resident application described below, allowing data generated by these applications to be associated with user interface elements.

[0405] The process of sub-step **2925** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of sub-step **2925** begins at sub-step **3801** where the flags associated with the selected user interface element **154** are set for output. At the sub-step **3803** if the data associated with the selected user interface element is present and identifies a valid delegator descriptor then the process proceeds to sub-step **3805**. Otherwise, the process of sub-step **2925** concludes. At the sub-step **3805** a delegator header associated with the delegate command is appended to the output buffer. The process continues at the next sub-step **3807**, where if the application identified in the delegator header exists then the process continues at sub-step **3809**. Otherwise, the process of sub-step **2925** concludes. At sub-step **3809** the identified application is invoked by the

CPU **275** with a provided application protocol data unit. The process of sub-step **2925** continues at the next sub-step **3811** where if the operating system executing on the CPU card **100**B responded with success then the process continues at sub-step **3813**. Otherwise, the process of sub-step **2925** concludes. At sub-step **3813** the delegator response is appended to the output buffer and the process concludes.

[**0406**] 2.6.11 Output Object Data Process

[**0407**] **FIG. 39** is a flow diagram showing an output object data process as executed at sub-step **2817** when the CPU **275** of the CPU card **100**B outputs data to the reader **300**. The process of sub-step **2817** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The process of sub-step **2817** begins at sub-step **3001** where if an "encrypt outgoing data" flag associated with the user interface element **154** selected at step **2801** is set then the process **3000** continues at sub-step **3003**. Otherwise, the process **3000** proceeds directly to sub-step **3007**. At sub-step **3003** if there is a session key stored in the storage means **276**, as explained above with reference to Table 28, then the process of sub-step **2817** proceeds to sub-step **3005**. Otherwise, the process of sub-step **2817** proceeds to sub-step **3913** where a "no session key" error identifier is returned by the CPU **275** and the process of sub-step **2817** concludes. At sub- step **3005** outgoing data is encrypted. The process of sub-step **2817** continues at the next sub-step **3007** where the flags associated with the selected user interface element are transmitted to the reader **300** via the data contacts **278**. At the next sub-step **3009**, the length of the outgoing data is sent to the CPU **1045** of the reader **300**. The process of sub-step **2817** concludes at the next sub-step **3911** where the outgoing data is sent to the CPU **1045** of the reader **300**.

[**0408**] 2.6.12 Save State Process

[**0409**] **FIG. 40** is a flow diagram showing the save state process executed by the CPU **275** at sub-step **2611** of the proprietary instruction process of **FIG. 26** in response to a SAVE STATE command. The save state process saves all volatile state information to non-volatile memory of the storage means **276** and then generates a random state code of a specified length. The process of sub-step **2611** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As seen in **FIG. 40**, the save state process of sub-step **2611** begins at sub-step **4001** where the CPU **275** saves state information associated with a present state of the user interface card resident application to non-volatile memory of the storage means **276**. At the next sub-step **4003** the CPU **275** generates a random value to be used as a state code. The save state process of sub-step **2611** continues at the next sub-step **4005** where the state code is saved to the non-volatile memory of the storage means **276**. The process of sub-step **2611** concludes at the next sub-step **4007** where the state code is sent to the CPU **1045** of the reader **300** and can be stored in memory **1046**.

[**0410**] 2.6.12 Restore State Process

[**0411**] **FIG. 41** is a flow diagram showing the restore state process executed by the CPU **275** at sub-step **2603** of the proprietary instruction process of **FIG. 26** in response to a RESTORE STATE command. The restore state process is configured to erase previous state information stored in non-volatile memory of the storage means **276** if the presented state code is not identical to the previous state code generated during a previous SAVE STATE command. The process of sub-step **2603** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As seen in **FIG. 41**, the restore state process of sub-step **2603** begins at sub-step **4101** where if the present state code stored in non-volatile memory **276** matches the state code contained in the data field of the RESTORE STATE application protocol data unit then the process proceeds to sub-step **4103**. At sub-step **4103**, the state information stored in volatile memory of the storage means **276** is replaced by the previously stored state information stored in the non-volatile memory of the storage means **276**. Therefore, the save state process allows the user interface card resident application to return to the state that the application was in prior to the reader **300** being suspended to a low power state, for example, and it is not necessary for a user of the CPU card **100**B to log back into the CPU card **100**B. Such a function provides greater convenience to a user of the CPU card **100**B. At the next sub-step **4105**, the CPU **275** returns a successful operation identifier to the reader **300**.

[**0412**] If the presented state code is not correct, at sub-step **4101**, then the save state process of sub-step **2603** proceeds to sub-step **4107**, where the state information previously stored in non-volatile memory of the storage means **276** is erased. At the next sub-step **4109**, an operations failed error is returned by the CPU **275** to the CPU **1045** of the reader **300** and the process of sub-step **2603** concludes. Therefore, the user interface card resident application returns to an initial default state and a user of the CPU card **100**B is forced to log back in the CPU card **100**B, for example. This function provides the card **100**B with greater security since it prevents unauthorised readers from restoring the state information of the user interface card resident application and thus the CPU card **100**B. The state of the CPU card **100**B can only be restored when the present state code is equal to the previous state code supplied by the reader **300**.

2.6.13 Select File Process **FIG. 42** is a flow diagram showing a select file process executed by the CPU **275** at sub-step **2703** in response to a SELECT FILE command being received by the user interface card resident application executing within the CPU **275**. The process of sub-step **2703** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As seen in **FIG. 42** the process of sub-step **2703** begins at sub-step **4201** where if the file identified in the data field of the select file application protocol data unit (as shown in table 40) exists then the process continues at sub-step **4203**. Otherwise, a "no file" error is returned by the CPU **275** to the reader **300** and the process of sub-step **2703** concludes. At sub-step **4203**, if file control information was requested by the parameter P2 field of the select file application protocol data unit then the process continues at the next sub-step **4205**. Otherwise, the process of sub-step **2703** proceeds to sub-step **4209** where a success code is returned by the CPU **275** to the reader **300** and the process concludes.

At sub-step **4205**, file control information is sent by the CPU **275** to the CPU **1045** of the reader **300** and the process of sub-step **2703** concludes.

[**0413**] 2.6.14 Read Binary Process

[**0414**] **FIG. 43** is a flow diagram showing a read binary process executed by the CPU **275** at sub-step **2707** in response to a READ BINARY command being received by the user interface card resident application executing within the CPU **275**. The process of sub-step **2707** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. As shown in **FIG. 43**, the read binary process is executed to read data from a currently selected elementary file stored in the storage means **276**. The process of sub-step **2707** begins at sub-step **4301** where if the currently selected file is able to be read then the process proceeds to sub-step **4303**. Otherwise, the process proceeds to sub-step **4317** where a security error is returned to the reader **300** by the CPU **275**. At sub-step **4303** if the offset identified by the P1 and P2 parameters of the read binary application protocol data unit is valid then the process proceeds to sub-step **4305**. Otherwise, the process of sub-step **2707** proceeds to sub-step **4315** where a bad offset error identifier is returned to the reader **300** by the CPU **275**. At sub-step **4305** if the number of bytes requested by the Le field of the read binary application protocol data unit are able to be read by the CPU **275**, then the process proceeds to sub-step **4307**. Otherwise, the process proceeds to sub-step **4309**. At sub-step **4307**, all of the bytes of the requested file are sent to the reader **300** by the CPU **275**. The process continues at sub-step **4311** where a success code identifier is returned to the reader **300** by the CPU **275** and the process of sub-step **2707** concludes. At sub-step **4309**, all available bytes of the requested file are sent to the user interface card resident application by the CPU **275**. At the next sub-step **4313** an end of file warning is returned by the CPU **275** and the process of sub-step **2707** concludes.

[**0415**] 2.6.15 Write Binary Process

[**0416**] **FIG. 44** is a flow diagram showing a write binary process executed by the CPU **275** at sub-step **2711** in response to a WRITE BINARY command being received by the user interface card resident application executing within the CPU **275**. The process of sub-step **2711** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The write binary process is executed at sub-step **2711** to write data to a currently selected elementary file. As seen in **FIG. 44**, the process of sub-step **2711** begins at sub-step **4401** where if the currently selected file is able to be written to by the CPU **275** then the process proceeds to sub-step **4403**. Otherwise, the process proceeds to sub-step **4417** where a security error identifier is returned to the reader **300** by the CPU **275**. At sub-step **4403** if the offset identified by the P1 and P2 parameters of the write binary application protocol data unit are less than the maximum length of the selected file then the process of sub-step **2711** proceeds to sub-step **4405**. Otherwise, the process of sub-step **2711** proceeds to sub-step **4415** where a bad offset error identifier is returned to the reader **300** by the CPU **275**. At sub-step **4405** if the number of bytes requested by the data field of the write binary application protocol data unit are able to be written by the CPU **275**, then the process proceeds to sub-step **4407**. Otherwise, the process proceeds to sub-step **4409**. At sub-step **4407** all of the bytes of the requested file are written to by the CPU **275**. The process continues at sub-step **4411** where a success code identifier is returned to the reader **300** by the CPU **275** and the process of sub-step **2711** concludes. At sub-step **4409**, all available bytes of the requested file are written by the CPU **275** to the selected file stored in the storage means **276**. At the next sub-step **4413** an end of file warning identifier is returned by the CPU **275** to the reader **300** and the process of sub-step **2711** concludes.

[**0417**] 2.6.16 Erase Binary Process

[**0418**] **FIG. 45** is a flow diagram showing an erase binary process executed by the CPU **275** at sub-step **2715** in response to an ERASE BINARY command being received by the user interface resident application executing within the CPU **275**. The process of sub-step **2715** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The erase binary process is executed at sub-step **2715** to truncate a currently selected elementary file stored in the storage means **276**, depending on the parameters received with the ERASE BINARY command. The process of sub-step **2715** begins at sub-step **4501** where if the write permission of the currently selected file is valid then the process proceeds to sub-step **4503**. Otherwise, the process proceeds to sub-step **4517** where a security error is returned to the reader **300** by the CPU **275**. At sub-step **4503** if the desired length of the file identified by the offsets of the P1 and P2 parameters of the erase binary application protocol data unit is less than the current length of the file then the process proceeds to sub-step **4505**. Otherwise, the process of sub-step **2715** proceeds to sub-step **4513** where a bad offset error identifier is returned to the reader **300** by the CPU **275**. At sub-step **4505** if the selected file is erasable (i.e. it is not write protected and an associated 'erasable' flag is set) then the process proceeds to sub-step **4507**. Otherwise, the process proceeds to sub-step **4511**. At sub-step **4507** all of the requested bytes are erased from the selected file by the CPU **275**. The process continues at sub-step **4509** where a success code identifier is returned to the reader **300** by the CPU **275** and the process of sub-step **2715** concludes. At sub-step **4511**, a bad file warning is returned to the reader **300** by the CPU **275** and the process of sub-step **2715** concludes.

[**0419**] 2.6.17 Get Challenge Instruction

[**0420**] 2.6.17.1 Get Challenge

[**0421**] **FIG. 46** is a flow diagram showing a get challenge process executed by the CPU **275** at sub-step **2719** in response to a GET CHALLENGE command being received by the user interface card resident application executing within the CPU **275**. The process of sub-step **2719** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The get challenge process is executed to provide challenge data in order to allow the security level of the CPU card **100**B to be changed. The process of sub-step **2719** begins at sub-step **4601** where the CPU **275** generates a random data string. At the next step **4603** the data string is

saved to the storage means **276**. Then at sub-step **4605** the retry counter stored in the storage means **276** is set to maximum. The process of sub-step **2719** concludes at sub-step **4607** where the generated data string is sent to the CPU **1045** of the reader **300**.

[**0422**] 2.6.17.2 External Authenticate

[**0423**] **FIG. 47** is a flow diagram showing an external authenticate process executed by the CPU **275** at sub-step **2723** in response to a EXTERNAL AUTHENTICATE command being received by the user interface card resident application executing within the CPU **275**. The process of sub-step **2723** is preferably implemented as part of the user interface card resident application program executing within the CPU **275** of the microprocessor **259** and being stored in the storage means **276**. The external authenticate process decrypts presented data using a private key associated with a requested security level. The process of sub-step **2723** begins sub-step **4701** where if the requested security level is lower than the current security level then the process proceeds sub-step **4703**. Otherwise, the process proceeds to **4709**. At sub-step **4703** the security level of the CPU card **100**B is set to the requested security level by the CPU **275**. The process of sub-step **2723** continues at the next sub-step **4705** where the retry counter stored in the storage means **276** is reset. At the next sub-step **4707** a success code identifier is returned by the CPU **275**.

[**0424**] At sub-step **4709** if there is any command data associated with the EXTERNAL AUTHENTICATE command then the process proceeds to sub-step **4711**. Otherwise, the process proceeds to sub-step **4719** where a bad command data error identifier is returned by the CPU **275**. At sub-step **4711** if the retry counter is greater than zero then process proceeds to sub-step **4713**. Otherwise, the process proceeds to sub-step **4721** where a bad challenge error identifier is returned to the reader **300** by the CPU **275**. At sub-step **4713** if a private key exists for the requested security level then the process proceeds to sub-step **4715**. Otherwise, the process proceeds to sub-step **4723** where a data not found error is returned to the reader **300** by the CPU **275**. At sub-step **4715**, the command data associated with the external authenticate command is decrypted using the private key. The process of sub-step **2723** continues at the next sub-step **4717** where if the decrypted data matches the original challenge data then the process proceeds to sub-step **4703**. Otherwise, the process of step **2723** proceeds to sub-step **4725** where the retry count is decremented. The process of sub-step **2723** concludes at the next sub-step **4727** where the CPU **275** generates the number of attempts remaining.

[**0425**] The aforementioned preferred methods comprise a particular control flow. There are many other variants of the preferred methods which use different control flows without departing the spirit or scope of the invention. Furthermore one or more of the sub-steps of the preferred method(s) may be executed in parallel rather sequential.

[**0426**] The foregoing describes only some embodiments of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiments being illustrative and not restrictive. For example, the user interface elements **114**, **154** can be positioned otherwise than on the smart card **100**. The user interface elements **114**, **154** may, for example, be displayed on a display device (e.g. **616**).

The claims defining the invention are as follows:

1. A program for allowing access to data associated with at least one file stored within an electronic card having one or more associated user interface elements, said electronic card comprising a card portion having an electronic apparatus attached thereto, said electronic apparatus comprising a processor means coupled to a memory means, said processor means being adapted for coupling to a reading device to facilitate reading of said electronic card, said memory means being configured to retain said files and said program for execution by said processor means, said program comprising:

code for relating reading signals generated from a user selection of at least one of said associated user interface elements with a first file stored within said memory, said first file being classified by said program as protected; and

code for allowing access to data of a second file associated with said first file, said second file being classified by said program as unprotected.

2. The program according to claim 1, wherein data associated with said protected files is inaccessible.

3. The program according to claim 1, wherein said files are assigned one of a plurality of levels of protection.

4. The program according claim 1, wherein said association is a file identifier.

5. The program according to claim 1, wherein said reading signals are provided by a substantially transparent touch sensitive membrane of said reading device arranged to overlay said electronic card, said reading signals being received from the reading device upon a press on said touch sensitive membrane when said electronic card is inserted into said reading device.

6. The program according to claim 5, wherein said relating said reading signals comprises comparing said reading signals to data of said first file.

7. The program according to claim 1, further comprising code for buffering said reading signals associated with said selected user interface element in an input buffer configured within said memory.

8. The program according to claim 1, wherein said associated user interface elements are formed on a surface of said card portion.

9. The program according to claim 1, wherein said associated user interface elements are formed remotely to said card portion.

10. The program according to claim 1, wherein at least one further file associated with said first file is classified by said program as protected.

11. A method of allowing access to data associated with at least one file stored within an electronic card having one or more associated user interface elements, said electronic card comprising a card portion having an electronic apparatus attached thereto, said electronic apparatus comprising a processor means coupled to a memory means, said processor means being adapted for coupling to a reading device to facilitate reading of said electronic card, said memory means being configured to retain at least said files, said method comprising the steps of:

relating reading signals generated from a user selection of at least one of said associated user interface elements with a first file stored within said memory, said first file being classified as protected; and

allowing access to data of a second file associated with said first file, said second file being classified as unprotected.

12. The method according to claim 11, wherein data associated with said protected files is inaccessible.

13. The method according to claim 11, wherein said files are assigned one of a plurality of levels of protection.

14. The method according claim 11, wherein said association is a file identifier.

15. The method according to claim 11, wherein said reading signals are provided by a substantially transparent touch sensitive membrane of said reading device arranged to overlay said electronic card, said reading signals being received from the reading device upon a press on said touch sensitive membrane when said electronic card is inserted into said reading device.

16. The method according to claim 15, wherein said relating said reading signals comprises comparing said reading signals to data of said first file.

17. The method according to claim 11, further comprising the step of buffering said reading signals associated with said selected user interface element in an input buffer configured within said memory.

18. The method according to claim 11, wherein said associated user interface elements are formed on a surface of said card portion.

19. The method according to claim 11, wherein said associated user interface elements are formed remotely to said card portion.

20. The method according to claim 11, wherein at least one further file associated with said first file is classified by said program as protected.

21. An apparatus for allowing access to data associated with at least one file stored within an electronic card having one or more associated user interface elements, said electronic card comprising a card portion having an electronic apparatus attached thereto, said electronic apparatus comprising a processor means coupled to a memory means, said processor means being adapted for coupling to a reading device to facilitate reading of said electronic card, said memory means being configured to retain at least said files, said apparatus comprising:

relating means for relating reading signals generated from a user selection of at least one of said associated user interface elements with a first file stored within said memory, said first file being classified as protected; and

access allowing means for allowing access to data of a second file associated with said first file, said second file being classified as unprotected.

* * * * *